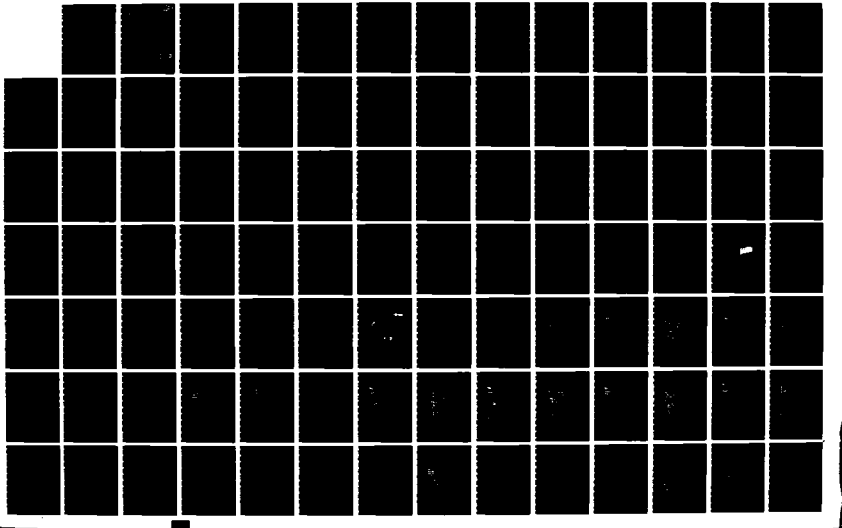
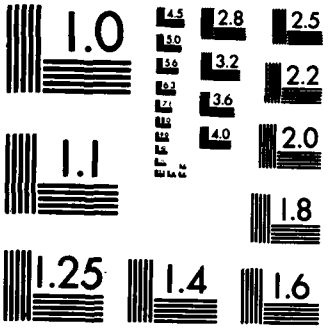
AD-A136 850 AUTOMATED EN ROUTE AIR TRAFFIC CONTROL ALGORITHMIC SPECIFICATIONS VOLUME 2.. (U) FEDERAL AVIATION ADMINISTRATION WASHINGTON DC SYSTEMS ENGINEE.. 1/2

UNCLASSIFIED J A KINGSBURY ET AL. 31 SEP 83 F/G 9/2 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

U.S Department
of Transportation
Federal Aviation
Administration
Office of Systems
Engineering Management
Washington, D.C. 20591

Automated En Route Air Traffic Control
Algorithmic Specifications

AIRSPACE PROBE

Volume 2

September 1983

Report No. DOT/FAA/ES-83/5

This document is available to the
U.S. public through the
National Technical Information Service,
Springfield, Virginia 22161

Technical Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
DOT/FAA/ES-83/5	AD-A136 250	

4. Title and Subtitle	5. Report Date
Automated En Route Air Traffic Control Algorithmic Specifications AIRSPACE PROBE Volume 2	September 31, 1983
	6. Performing Organization Code AES-320
7. Author(s) J.A. Kingsbury, N.S. Malthouse K.B. Schwamb	8. Performing Organization Report No. FAA-ES-83/5
9. Performing Organization Name and Address Systems Engineering Service Department of Transportation Federal Aviation Administration 800 Independence Ave., S.W., Washington, D.C. 20591	10. Work Unit No. (TRAIS)
	11. Contract or Grant No.
12. Sponsoring Agency Name and Address Same as #9 above.	13. Type of Report and Period Covered
	14. Sponsoring Agency Code AES

15. Supplementary Notes

16. Abstract

This Algorithmic Specification establishes the design criteria for four advanced automation software functions to be included in the initial software package of the Advanced Automation System (AAS). The need for each function is discussed within the context of the existing National Airspace System (NAS). A top-down definition of each function is provided with descriptions on increasingly more detailed levels. The final, most detailed description of each function identifies the data flows and transformations taking place within each function.

This document consists of five volumes. Volume 2, Airspace Probe, contains a functional design for the use of trajectory data to predict penetrations of airspace volumes from which the general flying public is normally restricted.

The other four volumes of this specification provide design criteria for the following:

- o Volume 1, Trajectory Estimation
- o Volume 3, Flight Plan Conflict Probe
- o Volume 4, Sector Workload Probe
- o Volume 5, Data Specification

17. Key Words	18. Distribution Statement
Automation, Air Traffic Control, Automated Decision Making, En Route Traffic Control, Artificial Intelligence, Advanced Automation System	Document is available to the U.S. public through the National Technical Information Service, Springfield, VA 22161

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified		

Form DOT F 1700.7 (8-72) Reproduction of completed page authorized

EXECUTIVE SUMMARY

This specification establishes design criteria for an Airspace Probe algorithm, part of the initial automation for the advanced automation system of the Federal Aviation Administration's (FAA's) Air Traffic Control (ATC) system. The algorithm provides data to construct a message to air traffic controllers when an aircraft is predicted to get too close to terrain or other areas wherein flight is restricted.

Airspace Probe is designed to be compatible with current air traffic control procedures and its design is an extension of the Enroute Minimum Safe Altitude Warning function of NAS Stage A. Airspace Probe extends the geographical coverage by providing a warning for controllers if an aircraft flight plan penetrates Enroute Minimum Safe Altitude Warning areas or Special-Use Airspaces. Airspace Probe also extends the time over which a warning may occur by using the flight plan to predict penetrations.

Airspace Probe algorithms assume that each airspace area is represented by a polygonal volume. The geographical coordinates, activation and deactivation times, and a maximum and minimum altitude have been provided by adaptation or supervisor interaction. After boundaries are defined, the Airspace Probe algorithm automatically detects penetrations of these areas. It processes aircraft trajectories which are derived from ATC approved flight plans for aircraft operating within an Instrument Flight Rule (IFR) context. The trajectory is checked to see if it intersects any Enroute Minimum Safe Altitude Warning areas or Special-Use Airspaces in the Planning Region. If any intersections are found, data describing the penetrations are stored in the data base. The Airspace Probe is invoked automatically when an incoming aircraft's flight plan is received by a center, when an aircraft's flight plan is amended and when flight plans are resynchronized. When any of these things occur, trajectories are reprobed to account for the change. If a supervisor activates and deactivates an area, the trajectories are also reprobed to incorporate this change.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

TABLE OF CONTENTS
(Concluded)

LIST OF ILLUSTRATIONS

1. INTRODUCTION

The Federal Aviation Administration (FAA) is currently in the process of developing a new computer system, called the Advanced Automation System (AAS), to help control the nation's air traffic. The AAS will consist of new or enhanced hardware (i.e., Central Processing Units, memories, and terminals) and new software.

The new software will retain most or all of the functions in the existing National Airspace System (NAS) En Route Stage A software. The algorithms will need to be coded and, in some cases, revised. In addition, the new AAS software will contain several new functions that make greater use of the capabilities of automation for Air Traffic Control (ATC). When fully implemented, these new functions are intended to detect and resolve many routine ATC problems.

The initial implementation of the AAS, described in the AAS Specification [1], will provide the ability to detect some common ATC problems. To meet the requirements of the AAS, several new ATC functions need to be postulated and described. Four of these functions are described in this document: Trajectory Estimation, Flight Plan Conflict Probe, Airspace Probe, and Sector Workload Probe [Volumes 1, 2, 3, and 4]. Together, they represent an initial level of automation and the beginnings of the evolution of the ATC system in accordance with the NAS Plan [2]. The NAS Plan represents an overview of the complete set of changes proposed to NAS in the coming decade.

1.1 Purpose

The purpose of this volume is to identify design criteria for Airspace Probe. Airspace Probe is one of the advanced automation functions called for in the AAS Specification. The design criteria specified in this volume are based on the existing NAS and the specification of the AAS. The AAS specification describes the Airspace Probe function and proposed some high-level requirements for this function.

1.2 Scope

This algorithmic specification presents design criteria for a computational framework of Airspace Probe. The framework is a set of algorithms which collectively describe how it may be possible to detect aircraft that are in danger of violating certain separation standards with given airspace volumes where

normal flight is restricted. It may be viewed as a candidate for consideration in the final design. However, it is not intended to be the complete final design for Airspace Probe in the AAS.

The framework establishes the requirments for input and output data and provides a description of the flow of control of data as it is transferred from input to output. Some of the principal requirements have been identified in the "Operational and Functional Description of AERA 1.01" [3]. To the extent possible, the data are discussed using existing NAS terminology.

1.3 Organization of This Document

The remainder of Section 1 provides a description of Airspace Probe's role in the larger ATC context and in future enhancements of the ATC System. Both the operational considerations and processing methods of Airspace Probe are summarized. Section 2 defines the terminology used in the specification and discusses the factors which influence the design of the algorithms.

Descriptions of the algorithms are contained in Section 3, Airspace Probe Functional Design, and in Section 4, Detailed Description. The Airspace Probe function, like the other advanced automation functions, is divided hierarchically (from top to bottom) into subfunctions, components, and elements (underlined words in Sections 1 and 2 are critical to the understanding of this specification and their definitions can be found in the Glossary, Appendix D). Section 3 specifies the design, environment, and assumptions of the subfunctions (e.g., the First-Order Coarse Filter), and outlines their components (e.g., Grid Chain Generation). Section 4 provides a detailed description of each subfunction's components, including their mission, data requirements, and processing details, and in some cases includes a discussion of a component's elements.

Appendix A defines the data shared by the various subfunctions of Airspace Probe. (Similarly, Volume 5 of this document contains the global data shared by the functions defined in Volumes 1 through 4.) Appendix B provides a description of several elements used in several places in Section 4. Appendix C provides mathematical derivations of certain formulas used in the specification. Supplementary information concerning polygon penetration computations is provided. Appendix D, as mentioned above, contains a glossary of those terms that are critical to an understanding of this specification.

A Program Design Language (PDL) which describes high-level control logic using structured English is used as needed to describe the algorithms in this specification. A description of this PDL is contained in Appendix E. Finally, Appendix F provides a complete list of references.

1.4 Role of Airspace Probe in the Overall Air Traffic Control System

The Airspace Probe algorithm evolves from the functions of the current Air Traffic Control System and the needs of the future Air Traffic Control System as given in the FAA's National Airspace System Plan [2,4].

1.4.1 System Context

The Continental United States airspace is partitioned among 20 centers or Air Route Traffic Control Centers (ARTCCs). The ARTCCs control regions bounded horizontally by polygons that stretch vertically from the center floor to 60,000 feet. Each center's airspace is further divided into areas, which are in turn divided into sectors. Areas and sectors are polygonal regions with floors (either a specified altitude or the center floor), and ceilings. The sectors of each area are staffed by a group of air traffic controllers (or controllers) specifically trained for that area.

In the current ATC system, pilots decide their desired means to reach their destination consistent with current navigational and ATC practices. This intent is then filed with the ATC system as a flight plan and approved as filed or altered by ATC for operating under Instrument Flight Rules (IFR). Alternatively, flight plans that are executed daily or on a regularly scheduled basis reside in a data base and are retrieved automatically unless altered or suspended. A flight plan modification may be initiated by a controller or the pilot. Advanced automation functions of the AAS can deal only with those aircraft filing IFR flight plans.

Controllers are responsible for monitoring flights as they pass through their sectors and for helping pilots achieve their objectives. They watch a block of symbols representing the aircraft's radar track position as it moves across a display console; the aircraft's identity, altitude, and other information are also displayed. Controllers institute control actions as needed to perform such functions as helping pilots avoid close approaches with other aircraft, honoring pilot requests for new routes, rerouting flights to avoid special airspaces

or severe weather, and queuing aircraft into the major terminal areas.

1.4.1.2 Need for Airspace Probe

The FAA has developed an automated tool for the controller, En Route Minimum Safe Altitude Warning (E-MSAW), to assist in detection of penetration of restricted flight airspaces. In that function, aircraft track positions and velocities are compared to the coordinates of terrain obstructions to determine if penetrations of minimum safe altitude could occur. The controller receives a displayed warning upon algorithmic detection of an imminent penetration of minimum safe altitude standards. E-MSAW provides the controller with an alert for potentially dangerous situations where aircraft might get too close to terrain obstructions (natural or man-made). As long as pilots stay on published routes, controllers need only short-term warnings when flights stray too close to terrain or volumes wherein general flight is restricted. Pilots filing published routes are provided with both minimum altitude requirements and the assurance that no published route penetrates a restricted flight regime. A flight violating published altitude requirements or penetrating a restricted area implies the need for "blunder" detection for the controller. Such a detection device is not a strategic prediction of problems.

With the increase in the use of unrestricted, user-preferred routes expected as the advancing level of automation allows, pilots will run the risk of unintentionally filing too close to restricted flight airspaces. Controllers need more efficient long-term warnings for penetrations predicted for this growing class of flyers.

The Airspace Probe is an extension of the E-MSAW concept. Airspace Probe can alert the controller long periods in advance of any projected penetration of pertinent airspace volumes. It uses an ATC-derived aircraft trajectory rather than track information. Airspace Probe provides for an alert not only for E-MSAW areas but for other areas as well. These could include NAS-adapted Restricted Areas and Warning Areas, Military Operations Areas, and other Special-Use Airspaces. The alert can then lead to a resolution of the penetration far in advance of projected entry time, thus helping to avoid inefficient maneuvers while facilitating greater use of user-preferred routes.

1.4.2 Role of Airspace Probe in Future System Enhancements

In the initial version of the Advanced Automation System [1], the Airspace Probe will be only a detection service which provides results for a manual resolution process. Later, results will feed into an automatic resolution service. As initially conceived, the Airspace Probe detects conflicts, the display generation functions are responsible for gathering information for the controller and displaying that information, and the controller plans resolution maneuvers for the aircraft. In a scenario of the evolution of ATC automation [5], future plans provide for continuing the current strategic detection service and decreasing the controller's responsibility for generating resolution maneuvers. This may be done by allowing the controller to choose from a ranked list of alternative resolutions or by providing the automatic resolution service itself.

Future automation plans also provide that Airspace Probe and related functions will predict and resolve penetrations with an enhanced set of geographic areas and include a mechanism for strategic conflict detection and resolution for dynamic areas (such as weather), as well.

1.5 Airspace Probe Summary

The Airspace Probe provides an aid for controllers to determine if an aircraft flight plan penetrates designated areas called "Minimum Safe Altitude Warning Areas" and "Special-Use Airspaces." Special-Use Airspaces are defined in the Airman's Information Manual [6]. These include, but are not limited to, Restricted Areas, Warning Areas, Prohibited Areas, and Military Operations Areas. Each aircraft's planned route of flight is compared against all these areas to check for intersections or penetrations. If a penetration is found, the identity of the area and the penetration coordinates are saved for retrieval and display as appropriate by the display functions.

1.5.1 Operational Description

Airspace Probe operates within the context of the AAS [1]. Other functions separate from Airspace Probe provide Airspace Probe with the environmental data needed to predict penetrations of certain airspaces. These data are discussed in adaptation guidelines [7]. Adaptation is that process of collecting important, relatively static environmental data and storing them in system-accessible data bases. Included in such

data are the geographical boundaries of the volumes of airspace which are used by Airspace Probe.

From a controller's point of view, Airspace Probe (in combination with the display generating functions, Situation Display, and Trajectory Estimation) provides information to help detect penetrations of special airspaces. The Airspace Probe function uses data describing the Special-Use Airspaces and E-MSAW areas and maintains the data describing any penetrations predicted. When a penetration is detected between an aircraft trajectory and a Special-Use Airspace or E-MSAW area, data for a controller display is updated. This operation is described in more detail by Swedish et al. [3]. The displays may provide such details of the penetrations as:

- Aircraft involved
- Location
- Conflict type
- Time to conflict
- Graphical display of conflict

From this information, the controller may develop a tentative resolution approach such as amending the flight plan. This may be done in the context of the Trial Plan Probe described operationally by Swedish [3]. If a change in the flight plan is involved, the controller may receive probe results to make sure the tentative resolution resolves the penetration and does not create new ones. If the penetration is not resolved, the controller may try another tentative resolution. If the penetration is resolved, the flight plan change may be accepted (by the controller) and the flight plan data base is updated (in functions separate from Airspace Probe). The controller does not invoke Airspace Probe by itself but always in the context of a flight plan amendment. The controller has, at all times, the means to ask for the display of penetrations in a different form (i.e., graphical rather than textual).

1.5.2 Processing Overview

Data describing special airspaces are maintained in the data base by their x,y geographical coordinates. Other information about the area is also maintained such as the airspace identification, the minimum and maximum altitude, and the activation and deactivation times (where applicable). Polygons may be convex or may be mixed (with some concave angles). Area coordinates may only be changed in adaptation, but the area may be temporarily activated or deactivated by supervisor request.

Aircraft trajectories for IFR aircraft with valid flight plans are constructed by the Trajectory Estimation function. These trajectories are maintained as a series of points designating x,y (horizontal position), z (altitude) and t (time) at each point. Once these trajectories are available, then Airspace Probe can derive airspace penetration information.

Airspace Probe works in tandem with Trajectory Estimation: whenever the trajectory for an aircraft changes, Airspace Probe is automatically invoked to maintain the airspace penetrations data base. Airspace Probe compares the trajectory against all pertinent airspaces that are currently active using a series of progessively finer filters. The First-Order Coarse Filter and Second-Order Coarse process all polygons to accumulate candidate intersecting object polygons. The Fine Filter process this object polygon list to determine the intersection coordinates (if any). When trajectories intersect an area, a data structure which maintains information about the penetration is defined and stored in the data base. Any of the information maintained in the data base may be available for display to the controller.

2. DEFINITIONS AND DESIGN CONSIDERATIONS

Airspace Probe includes E-MSAW capabilities along with new capabilities. Inclusion of an extended set of airspace areas widens the responsibilities of Airspace Probe over that of E-MSAW, but the basic purpose remains unchanged and, so, the algorithms of Airspace Probe remain deeply rooted in the previous E-MSAW work.

This section introduces terminology used in this specification. Also provided is a set of design considerations which place Airspace Probe firmly within the AAS context.

2.1 System Design Definitions

Some terms introduced in Section 1 of this specification are of global interest across the AAS environment and include (in order of presentation):

1. Subfunction
2. Component
3. Element
4. Center
5. Areas
6. Sectors
7. Controllers
8. Flight Plan
9. Penetration
10. Adaptation

Other terms of interest only to Airspace Probe are introduced below.

2.1.1 Airspace Types

Special-Use Airspaces are areas wherein aircraft operations are limited. This section lists and defines the set of Special-Use Airspaces referenced in this specification. Airspace types are further defined in the Airman's Information Manual [6].

- Controlled Firing Areas

  Controlled Firing Areas are areas which contain activities which could be hazardous to nonparticipating aircraft. A unique feature of these areas is that activities are suspended if spotter aircraft, radar, or ground look-out positions indicate that a nonparticipating aircraft is approaching.

- Military Operations Areas

  Military Operations Areas (MOAs) consist of airspace defined by vertical and lateral limits which are established to separate military training activities from IFR traffic.

- Prohibited Areas

  Prohibited Areas are airspace volumes within which the flight of aircraft is prohibited. They contain airspace of defined dimensions identified by an area on the surface of the earth. These areas are established for security or other reasons associated with the national welfare.

- Restricted Areas

  Restricted Areas are airspace volumes within which the flight of aircraft is restricted. Aircraft activities within these areas must be confined because of the content of activities occurring in the area. Restricted areas denote the existence of unusual, often invisible hazards.

- Warning Areas

  Warning Areas are airspace beyond the three-mile limit over international waters which may contain hazards and should not be penetrated during periods of activity. Even though the activities in warning areas may be as hazardous as those in restricted areas, areas over international waters cannot be legally designated as restricted areas.

2.1.2 Modeling Environment Terms

A center represents a volume of airspace for air traffic control. Enclosing the center is the planning region. The boundary of the planning region is considered to be some horizontal distance outside that of the center: some 20 to 30 minutes of flying time in all directions.

Trajectory Estimation [Vol. 1] provides Airspace Probe with a trajectory for each aircraft with an IFR flight plan. A trajectory is a predicted path for the aircraft through the three spatial dimensions (x, y, z) and time. Each trajectory is conceptually a continuous, smooth curve in four dimensions.

However, trajectories are modeled as a series of lines (in space-time) called segments, joined together at their endpoints, called cusps. The data base provides trajectory information as a list of cusps:

$$\left\{C_i = (x, y, z, t)_i \mid i = 1, \ldots, n\right\}$$

The segments are the implied straight lines joining adjacent cusps. The trajectory is the ordered sequence of these segments.

It is convenient for purposes of Airspace Probe to enclose the horizontal extent of the planning region in a grid. The grid covers the planning region with squares, called cells, aligned with the x,y coordinate axes of the coordinate system used by Trajectory Estimation. These cells provide a reference for geographical features in terms of their location within a numbered cell.

The grid structure associated with E-MSAW is the underlying Radar Sort Box grid structure which is used primarily in Radar Data Processing. This grid structure was updated to incorporate E-MSAW information as described in NAS Stage A Automatic Tracking specification [8]. The requirements of Airspace Probe are satisfied by this grid structure. However, there is no guarantee that the AAS will incorporate the Radar Sort Box concept. Consequently, the remainder of this document refers to an Airspace Probe "grid" to give emphasis to the fact that a similar type of grid structure is necessary for Airspace Probe algorithms.

2.1.3 Airspace Probe Terms

Airspace Probe works with a trajectory and a set of airspace volumes. The trajectory is said to belong to the subject aircraft. The airspace volumes, which are assumed by Airspace Probe to be cross-referenced to the grid through adaptation, form the set of object polygons.

The Airspace Probe algorithm is executed through a sequence of filters. A filter is a logical subalgorithm the input of which is a subset of all object polygons and the output of which is a subset of the input. Input to the first filter, called the First-Order Coarse Filter, is the entire object polygon set. Output from the last filter, called the Fine Filter, is the set of encounters. An encounter is an object polygon penetrated by the subject's trajectory. A nominee is an object polygon which is input to any filter except the First-Order Coarse Filter.

The subject's trajectory, upon initial processing in Airspace Probe, must be cross-referenced to the grid. In this process, the list of cells that the trajectory penetrates, called the grid-chain, is computed. The logical entity responsible for the cross-referencing is called the grid-chain generator.

2.2 Design Considerations

Environmental adaptation is assumed to record the identities, geometry and coordinates of all E-MSAW areas and Special-Use Airspaces (SUAs) that are physically within the planning region. The E-MSAW areas and SUAs are simple polygons in an (x,y) projection with flat tops and bottoms. The E-MSAW areas may cover the planning region giving an approximation to the geography and radar receiving capabilities of the underlying map. They all touch the ground and are under 25,500 feet in altitude. The other SUAs may be detached, floating above the planning region. The estimated population of protected airspaces is about 500 where most of them are E-MSAW polygons.

A typical planning region is a polygon with vertices established as latitude, longitude points. In environmental adaptation, the planning region is apportioned among multiple cells. Next, all E-MSAW areas and SUAs are positioned in the grid as shown in Figure 2-1. When adaptation is completed, each cell data element contains the identity of all polygons which intersect that cell. The opposite is also true. Each polygon data element adapted contains the list of grid cells the polygon intersects. Maintenance of both the polygon-by-cell and cell-by-polygon data bases is required to provide access to the cells when the polygons are activated or deactivated, and to provide access to the polygons when the cells enclosing the flight plan segment change.

The E-MSAW function which exists in NAS Stage A has been used as a source of some of the algorithms of Airspace Probe. The E-MSAW function has limited warning capabilities in comparison to those which have evolved for Airspace Probe. E-MSAW provides a tactical warning message to controllers when aircraft are too close to terrain obstructions. E-MSAW warns of imminent penetration of airspaces where "imminent" is defined to be less than five minutes into the future.

At the other end of the tactical-strategic spectrum, Airspace Probe provides information to construct a warning message to controllers when planned aircraft trajectories get too close to terrain and other Special-Use Airspaces. Using aircraft

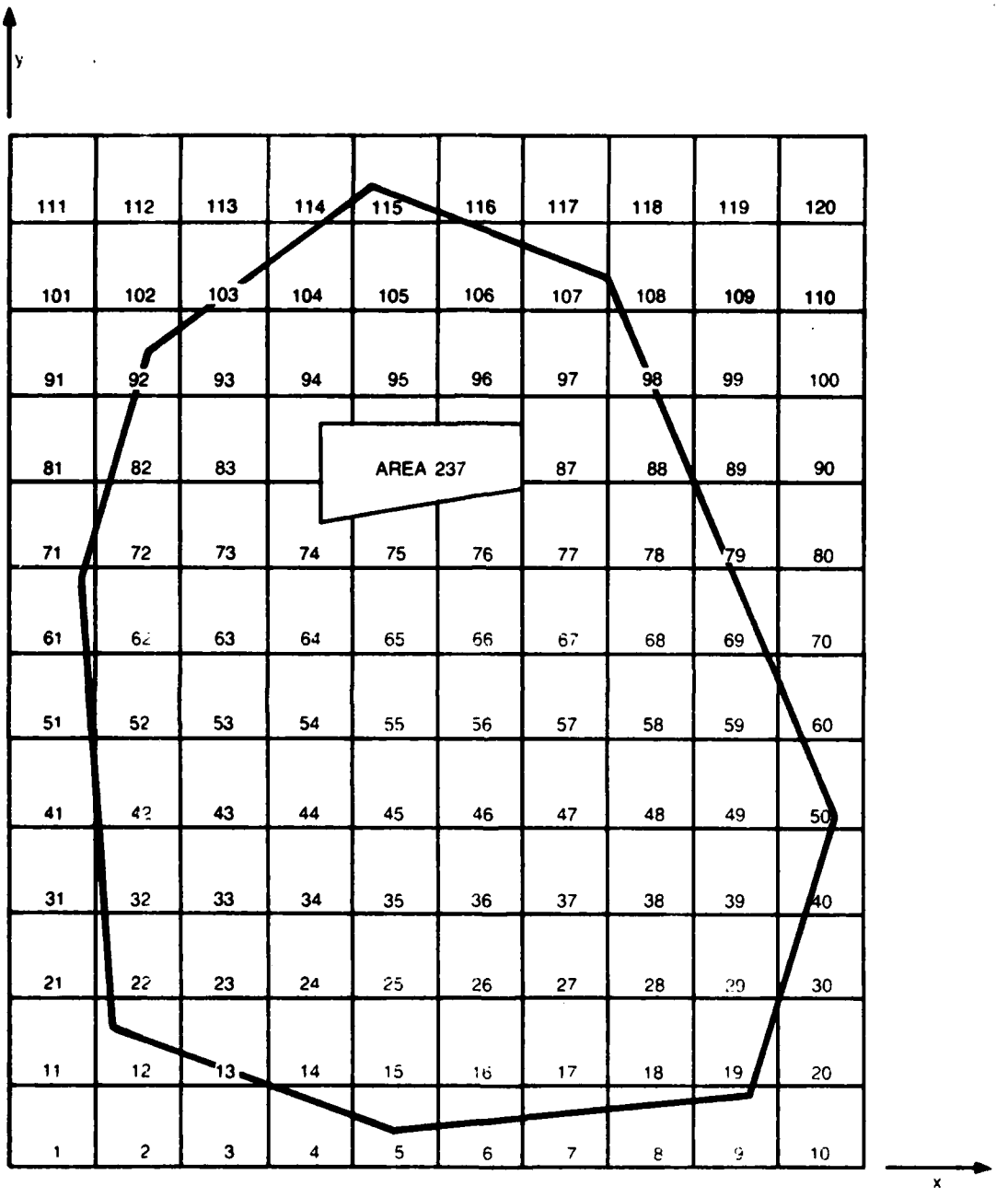


FIGURE 2-1
SPECIAL-USE AIRSPACE DEFINED ON
PLANNING REGION GRID

trajectories, Airspace Probe performs the same function without the temporal limitations.

The algorithm supporting the Airspace Probe has evolved from the E-MSAW algorithm [8,9]. The NAS Adaptation Process [7] provides the environmental data. Adaptation and the E-MSAW algorithm can be summarized as shown below:

- E-MSAW Area Adaptation:

  1. The airspace of the planning region is divided into a regular grid.

  2. The airspace terrain polygons are cross referenced with respect to the grid.

- E-MSAW Algorithm:

  1. The current position and velocity of the aircraft are projected ahead for some fixed time period (nominally two minutes) based on radar track data.

  2. The intersections between projected line segments and polygons are determined.

  3. The intersections are reported to the controller.

The two new features of Airspace Probe are incorporation of additional airspace volumes and the use of the aircraft trajectory for early penetration prediction. In addition, penetrations are maintained in the data base for display to the controller (either immediate or later display). The Airspace Probe algorithm works as shown below:

- E-MSAW Area and Special-Use Airspace Adaptation:

  1. The airspace of the planning region is divided into a regular grid.

  2. The E-MSAW areas and Special-Use Airspaces are cross-referenced with respect to the grid.

- Airspace Probe Algorithm:

  1. The planned aircraft trajectory is examined.

2. The intersections between planned trajectories and polygons are determined.

3. The intersections are stored in the data base.

3. AIRSPACE PROBE FUNCTIONAL DESIGN

This section identifies the environment in which Airspace Probe is to work in the AAS. The input and output data are identified along with activation sequences. At the end of this section, the major subfunctions of Airspace Probe are identified and a description of each subfunction is provided.

3.1 Environment

The prediction process of Airspace Probe uses the stored polygon information along with the predictions of future positions for aircraft from Trajectory Estimation to search for positions where an aircraft path (in four dimensions) penetrates an E-MSAW or Special-Use Airspace volume. Figure 3-1 depicts the Airspace Probe functional environment.

3.1.1 Input Data and Activation

The Airspace Probe function requires an initialized data base containing various types of data defining the environment. The environment is divided into a regular grid covering the entire x,y extent of the planning region. The (x,y,z,t) coordinates of E-MSAW and Special-Use Airspaces are input and cross-referenced to the grid.

Airspace Probe uses this environmental definition and data which specifies the trajectory to be probed. The algorithm typically processes one aircraft trajectory. In either case, the algorithm operates the same way. An aircraft is selected (separate from the Airspace Probe algorithm) and the trajectory is compared against the object polygons. A list of those polygons which intersect the aircraft trajectory is formed and data is stored describing the intersection.

3.1.1.1 Input Data

Airspace Probe requires input data through adaptation. Polygon adaptation ensures that the following data are accumulated which describe the E-MSAW and Special-Use Airspace environment:

- Grid specification
- Airspace polygon coordinates, (x,y,z,t), for each E-MSAW Area and Special-Use Airspace
- Polygons further defined in a polygon data base cross-referenced with the grid

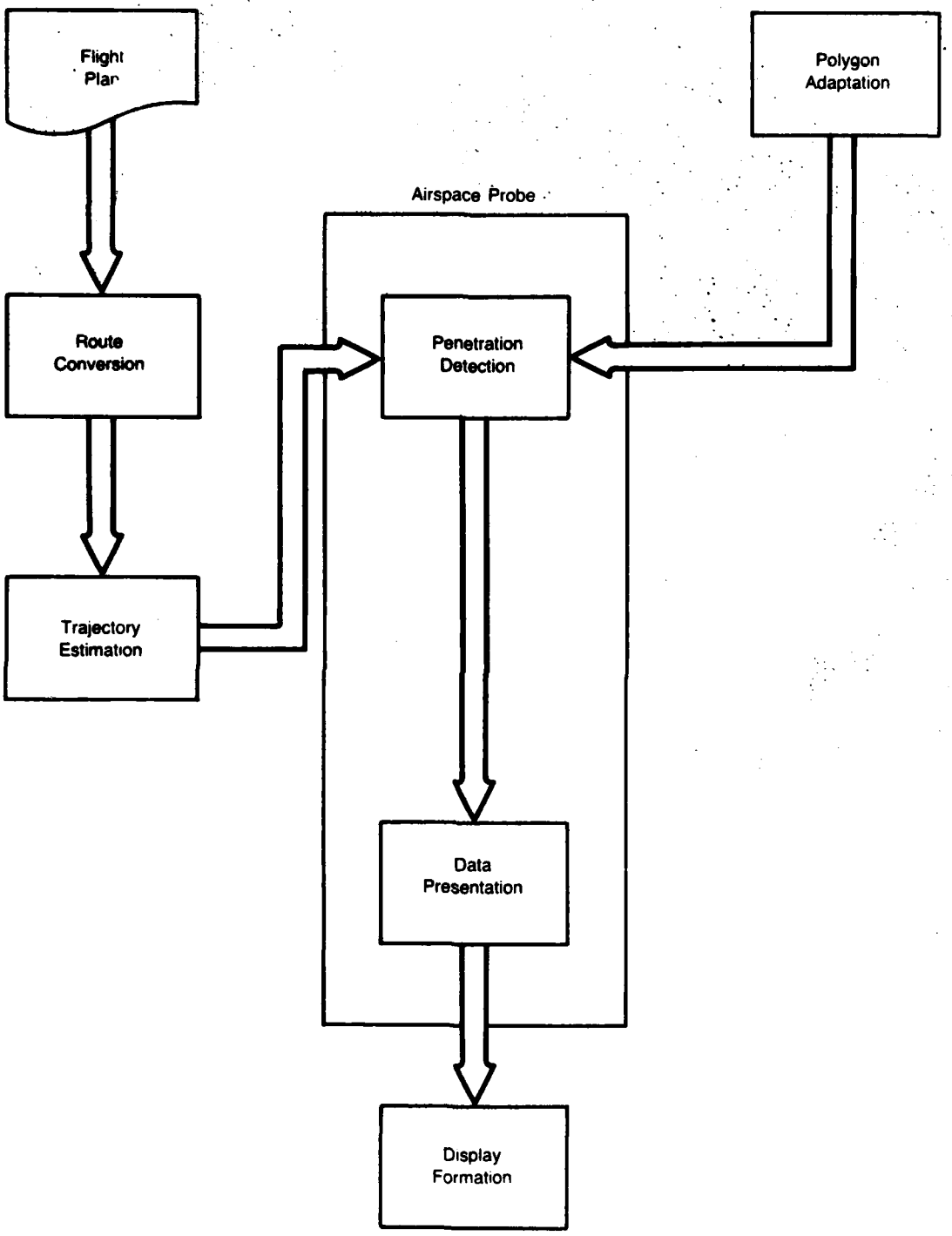


FIGURE 3-1
AIRSPACE PROBE FUNCTIONAL ENVIRONMENT

Airspace Probe must further be provided with an aircraft's trajectory which describes the path the aircraft is predicted to take through the planning region.

3.1.1.2 Automatic Activation Sequences

Airspace Probe may be initiated automatically to determine penetrations of protected airspace whenever the following events occur:

- The trajectory estimate for an aircraft changes. This could occur when a new aircraft enters the system, updates to trajectory time values are made, or a candidate plan is being examined by the controller. (See Section 3.1.1.3)

- The time bounds on any one Special-Use Airspace change through supervisory action. (See Section 3.1.1.4)

3.1.1.3 Controller Initiating Sequences

A controller may implicitly initiate Airspace Probe when he has used his strategic planning mechanism (i.e., Trial Plan Probe as described by Swedish [2]) to include some alteration in the aircraft's plan such as a change to the assigned altitude or speed. In these cases, Airspace Probe is invoked automatically. If the trajectory is not changed, however, the controller should not request Airspace Probe since no new information can be generated. He may only request more information about the penetrations already detected and stored.

3.1.1.4 Supervisor Activation and Deactivation

The supervisor may implicitly initiate the Airspace Probe when he activates or deactivates an area. In this case, the supervisor would change the time limits on a certain Special-Use Airspace. This action externally activates an Airspace Probe on a (possibly large) population of aircraft. The activation of Airspace Probe for each aircraft involved in this population is automatic. This activation sequence is not described further in this specification.

3.1.2 Output

The penetration detection algorithms of Airspace Probe identify encounters and store the data for use by the controller. Several types of data are stored (cf: Vol. 5, "Environmental Conflict").

- Polygon identification
- Aircraft identification
- Encounter time
- Encounter coordinates
- Advisory time

3.1.2.1 Information to the Controller

The Airspace Probe stores penetration information and makes it available for display by the display function. Any time a penetration between an aircraft trajectory and E-MSAW areas or Special-Use Airspaces is predicted, data for a controller display is updated. This data provides information about the penetrations of all aircraft into E-MSAW and Special-Use Airspace polygons such as:

- Aircraft identification
- Sector, grid, and polygon identification
- Penetration coordinates
- Time to penetrations

The display function is maintained as a separate entity. Thus, it has logic of its own to determine encounters eligible for display to the appropriate controller, select appropriate data to display, provide the desired display format, and choose the logical display on the appropriate logical device.

The display function sorts Airspace Probe encounter data by time and generates two types of warnings. If the time to penetration is more than X (system parameter) minutes, an advisory message is sent to the controller who is currently responsible for the aircraft. If the time to penetration is less than X (system parameter) minutes, an alert message is sent to the controller responsible at the position of penetration.

The display function selects appropriate data for display to the controller and provides the display format such as arrangement, choice of graphic or alphanumeric information, and (possibly) color of data items. In both the advisory and alert messages, the controller is presented with information required to identify the penetration and formulate a resolution. All information necessary to support the display function exists in the penetrations data base maintained by Airspace Probe.

3.1.2.2 Information to the Supervisor

When areas are activated or deactivated by the supervisor, no special information is provided from the initiation of Airspace Probe. However, the display functions should inform the supervisor that his request has been honored.

3.2 Design Assumptions

This section describes some assumptions made in the design of Airspace Probe algorithms. Of special importance are those assumptions placed on the context of the environmental data.

3.2.1 Polygon Adaptation

Adaptation of E-MSAW areas and Special-Use Airspace is assumed in this specification to provide the environmental information used by Airspace Probe algorithms. As in E-MSAW, the polygons are assumed to be cross-referenced to a grid where each polygon data element contains the identity of all the cells it intersects, and each cell data element contains the identity of all the polygons that intersect it. In particular, the following data are assumed:

- Cell data element (cf: Vol. 5, "Environmental_Cell")
  - cell identification
  - the polygon identification for each polygon intersecting this cell
- Polygon data element (cf: Vol. 5, Special_Use_Airspaces and E_MSAW_Areas)
  - polygon identification
  - cell identification for each cell this polygon intersects
  - airspace type (E-MSAW, etc.)
  - polygon type (convex, mixed)
  - list of (x,y) vertices for the polygon

- altitude extent of the polygon

- time extent of the polygon

The vertices of the polygon are assumed to be stored in a consistent ordering scheme: "clockwise" is used in this specification since that convention was adopted by E-MSAW. Furthermore, this specification assumes that the vertices stored for a polygon extends the real boundary of the polygon by a system parameter number of miles. This is necessary to account for the lateral positional uncertainty in a trajectory. This notion is portrayed in Figure 3-2.

3.2.2 Inherited E-MSAW Assumptions

Several major design assumptions are derived from the design of E-MSAW [8,9]:

- Special-Use Airspaces can be processed algorithmically like E-MSAW polygons are processed.

- It is not necessary to restrict Special-Use Airspace Polygons to convex polygons.

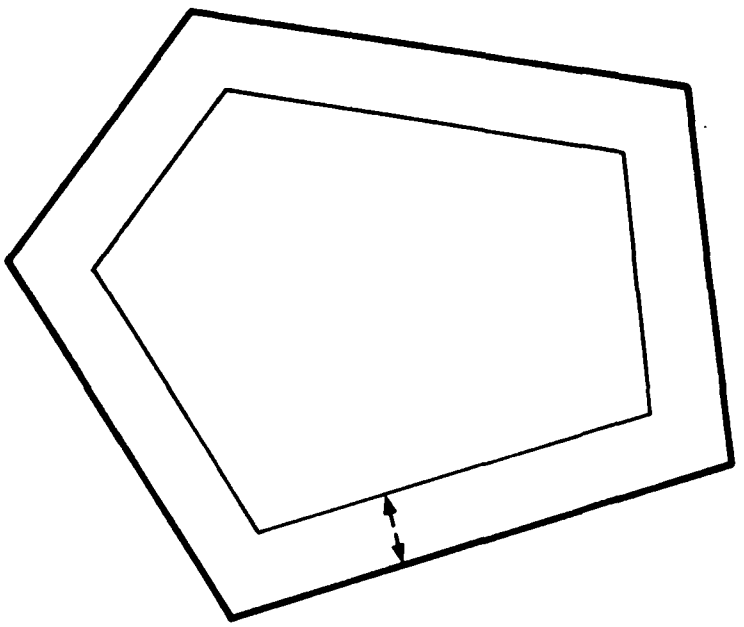
- When trajectories intersect a polygon several times, certain multiple intersections can be treated as one unique penetration.

3.3 Subfunctions

Three subfunctions to Airspace Probe are identified and described in this section. Each subfunction refines an input list of object polygons. At the termination of the Airspace Probe process, an output set of encounters is produced.

3.3.1 First-Order Coarse Filter

The First-Order Coarse Filter defines a nominee in terms of the x,y closeness of a polygon to a trajectory. The trajectory representing the aircraft's path is logically superimposed on the planning region grid and the grid-chain extracted. Polygons named in each cell of the confining grid-chain are added to the list of first level nominee polygons. Each such nominee has the property that the aircraft's trajectory intersects a grid cell the polygon also intersects. They are, therefore, "close" (relative to the grid). This process is shown in Figure 3-3.



Input Polygon

Expanded Polygon

FIGURE 3-2
EXPANDED POLYGON BOUNDARY
FOR AIRSPACE PROBE

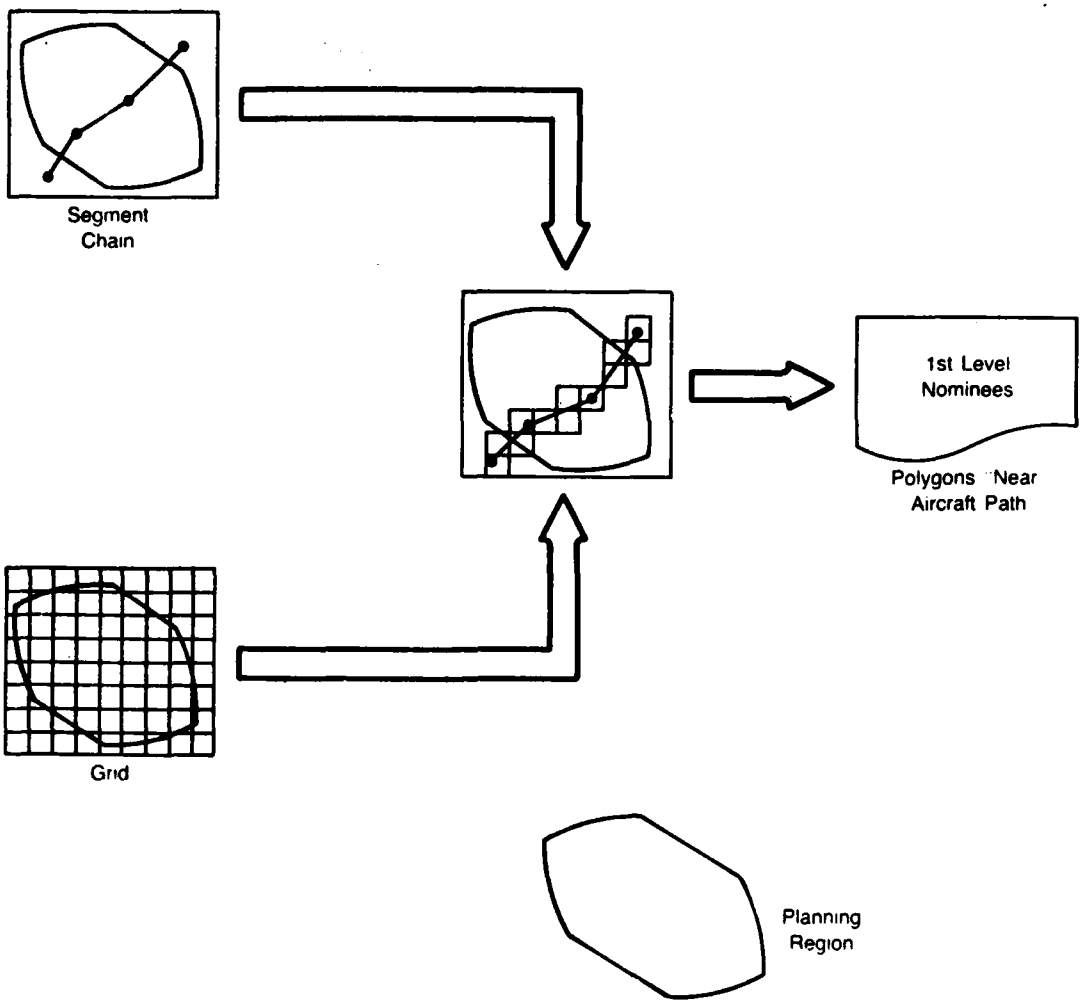


FIGURE 3-3
FIRST-ORDER COARSE FILTER SELECTION

3.3.2 Second-Order Coarse Filter

The Second-Order Coarse Filter defines a nominee in terms of an x,y,z,t closeness measure of a polygon to the trajectory. The polygons identified in the First-Order Coarse Filter are again compared to the trajectory. A series of interval intersection tests are performed between trajectory segments and one- and two-dimensional circumscribed right rectangles that envelop the polygon.

3.3.3 Fine Filter

The Fine Filter defines an encounter in terms of an exact intersection between the polygon and a trajectory segment. The polygons identified by the Second-Order Coarse Filter are again compared to the trajectory segment. Those polygons with the property that they intersect the aircraft trajectory are identified.

3.3.4 Encounter Processing

Encounter Processing stores information about the encounters identified by the Fine Filter. This data includes information such as the aircraft ID, route, altitude, time and position of penetration, and the identification of the Special-Use Airspace or E-MSAW area.

3.4 Extendability

Airspace Probe is expected to be enhanced in the future to predict penetrations of aircraft trajectories against weather polygons. This might be accomplished by generating a series of static polygons representing the weather cell at various times t-minutes apart, each with a lifetime of t-minutes or more. Such an extension requires no changes in the current algorithm. An alternative approach might define polygons to be dynamic in nature with an implied velocity vector and time extent. This dynamic nature would force changes in the Airspace Probe algorithm in two areas.

First, the moving polygon concept does not fit well with the grid structure serving the First-Order Coarse Filter. There is no temporal limit in the grid structure, itself, and a moving area would then cut a "swath" into the grid. For this reason, each moving polygon should not be incorporated into the Grid, but each moving polygon should automatically become a First-Order Nominee for every aircraft.

Second, the incorporation of moving polygons into the polygon population forces several upgrades in the execution of the Second-Order Coarse Filter and in the Fine Filter. The logic of these two entities can be easily changed to consider every polygon a dynamic polygon (with E-MSAW and Special-Use Airspaces having an assumed zero-velocity vector). A switch to a relative geometry (or aircraft centered) coordinate system can be made at the outset of processing, and the remainder of the filters executed as specified.

4. DETAILED DESCRIPTION

The penetration detection algorithms of Airspace Probe are arranged in a series of progressively more discriminating filters. Airspace Probe is composed of a First-Order Coarse Filter, a Second-Order Coarse Filter, a Fine Filter and an Encounter Processing routine. Polygons passing through all the filters are placed on a list of polygons which intersect the aircraft trajectory. Figure 4-1 illustrates the relationship of the components in the Airspace Probe.

4.1 First-Order Coarse Filter

4.1.1 Mission

The First-Order Coarse Filter for Airspace Probe is a mechanism for quickly selecting the proper subset of polygons (i.e., those which may intersect the aircraft trajectory) for further Airspace Probe processing. The inclusion of Minimum Safe Altitude Warning Areas into the population of polygons considered by Airspace Probe makes such a filter mandatory for reasons of efficiency. There can be, in the adaptation data base, several hundred Minimum Safe Altitude Warning Areas which can describe the topography of the underlying planning region. In fact, the whole planning region could be covered by such polygons.

The First-Order Coarse Filter of Airspace Probe is especially constructed to use stored (adapted) geographical information about the location of polygons and information from the trajectory of the aircraft to eliminate polygons on the basis of some *a priori* measure of closeness. Conceptually, if the path of the aircraft is contained entirely in the southern section of a planning region while a polygon is in the north, the polygon should be eliminated from further processing.

The selected polygons which are close to the aircraft path resulting from such a coarse filter should be a small subset of the total polygon population. That subset comprises a set of nominees. Even though the aircraft's path is close to the polygon, the path of the aircraft may or may not intersect the extent of a nominee polygon. Further Airspace Probe processing is necessary to determine the actual penetration status of the aircraft path with respect to each nominee.

```
ROUTINE Airspace_Probe;
PARAMETERS
   Loc_Fl_Id IN;
DEFINE VARIABLES
   Loc_Fl_Id          The identification of the aircraft being
                        probed for airspace conflicts ;.
##
   CALL First_Order_Coarse_Filter(Loc_Fl_Id IN);
   CALL Second_Order_Coarse_Filter;
   CALL Fine_Filter;
   CALL Encounter_Processing(Loc_Fl_Id IN);
END Airspace_Probe;
```

FIGURE 4-1
AIRSPACE_PROBE

4.1.2 Design Considerations and Component Environment

The First-Order Coarse Filter of Airspace Probe is designed to provide an efficient mechanism for examining an aircraft trajectory with respect to the airspace polygon environment. It uses an adapted grid structure to select a set of nominee polygons from the polygon population. These may intersect the trajectory of the aircraft. The complement of the set of nominees is a set of polygons which clearly do not intersect the trajectory. To perform its function, the First-Order Coarse Filter requires input defining the aircraft trajectory and input defining the environmental polygons cross-referenced to a grid structure. It produces output defining a list of Nominees.

The sequence of elements associated with the First-Order Coarse Filter is shown in Figure 4-2. Program design language is provided in this section for each element shown in Figure 4-2 with the exception of Grid and Linear_Discriminant_Classifier. A description of those two elements is provided in Appendix B.

Input Data

The input data required by the First-Order Coarse Filter consists of:

- System Global Data Base

  - TRAJECTORIES

    The aircraft's trajectory is obtained from the trajectories table using the flight identification input to the Airspace Probe algorithm.

  - VOLUMES

    The ceiling altitude of each airspace volume identified by the grid-chain generator is obtained for checking purposes.

  - ENVIRONMENTAL_CELL_CONTENTS

    A cell identified by the grid-chain generator is cross-referenced to each airspace polygon intersecting the cell. The identities of each polygon are retrieved for possible addition to the list of noninees.

```
First_Order_Coarse_Filter
   Cusps_To_Segments
   Grid_Chain_Generation
      Set_Up_Segment_Scan
         Grid
      Scan_Segment_To_Pick_Up_Cells
         Grid
         Add_Box
            Get_Lower_Left_Corner_Points
               Grid
            Linear_Discriminant_Classifier
```

FIGURE 4-2
ELEMENTS OF THE FIRST-ORDER COARSE FILTER

- ENVIRONMENTAL_GRID_PARAMETERS

  The nominal cell width is obtained.

- ENVIRONMENTAL_CELLS

  The extent of a cell is retrieved. In particular, the x and y extents are obtained to construct the boundary of the cell.

Output Data

The First-Order Coarse Filter produces a list of nominee polygons which must be processed through the remainder of the Airspace Probe algorithm.

- **Shared Local Data Base**

  - FIRST_ORDER_NOMINEES

    The identifies of the First-Order Coarse Filter Nominee polygons are stored in this table. These polygons must have the property that they intersect a cell that the aircraft's trajectory intersect and the ceiling altitude of the polygons are above the minimum altitude of the trajectory.

  - FL_CUSPS

    The trajectory of the aircraft is brought into local storage.

  - SEGMENTS

    The trajectory, which is a list of cusps, is arranged to yield an explicit line segment by line segment representation.

4.1.3 Component Design Logic

The Airspace Probe First-Order Coarse Filter is responsible for constructing a list of polygons known to be "close" to the route of the aircraft. The route of the aircraft is provided by the XYZT-Segments. Figure 4-3 provides a description of the control logic for the First-Order Coarse Filter. In the element Cusps_To_Segments (Figure 4-4), the trajectory of the aircraft is obtained and processed to yield the ordered set of segments which represents the aircraft's route.

```
ROUTINE First_Order_Coarse_Filter;
PARAMETERS
   Loc_Fl_Id IN;                    The Flight Identification
REFER TO GLOBAL
   TRAJECTORIES IN,
   VOLUMES IN;
REFER TO SHARED LOCAL
   FIRST_ORDER_NOMINEES OUT,
   FL_CUSPS OUT;
DEFINE TABLES
   GRID_CHAIN_VOLUMES      The volumes found in the grid chain
                              describing the aircraft trajectory
      volume_id            The volume identifier
      first_cusp_time      The first cusp before the grid chain
                              cell containing the volume
      all AGGREGATE (volume_id,first_cusp_time);
DEFINE VARIABLES
   Loc_Fl_Id               The Flight Identification
   Min_Fl_Z                The minimum altitude over the flight
   Ceiling_Altitude        The ceiling altitude of the polgon
                              being examined;
   ##
   FL_CUSPS = SELECT FIELDS time,x,y,z
      FROM TRAJECTORIES
      WHERE TRAJECTORIES.fl_id EQ Loc_Fl_Id
      ORDER BY TRAJECTORIES.time;
   CALL Cusps_To_Segments;
   CALL Grid_Chain_Generation (GRID_CHAIN_VOLUMES OUT);
   SELECT FIELDS z
      FROM FL_CUSPS
      INTO Min_Fl_Z
      WHERE FL_CUSPS.z EQ MIN(FL_CUSPS.z);
   REPEAT FOR EACH GRID_CHAIN_VOLUMES RECORD;
      SELECT FIELDS ceiling_altitude
         FROM VOLUMES
         INTO Ceiling_Altitude
         WHERE GRID_CHAIN_VOLUMES.volume_id EQ VOLUMES.volume_id;
      IF Min_Fl_Z LT Ceiling_Altitude
      THEN
         INSERT INTO FIRST_ORDER_NOMINEES
            (all = GRID_CHAIN_VOLUMES.all);
END First_Order_Coarse_Filter;
```

FIGURE 4-3
FIRST_ORDER_COARSE_FILTER

```
ROUTINE Cusps_To_Segments;
REFER TO SHARED LOCAL
   FL_CUSPS IN,
   SEGMENTS OUT;
DEFINE VARIABLES
   First_Cusp          The flag indicating that the first cusp of
                          the trajectory is being processed
   Previous_Time       The time of the previous cusp;
##
   First_Cusp = "true";
   REPEAT FOR EACH FL_CUSPS RECORD;
      IF First_Cusp EQ "true"
      THEN
         INSERT INTO SEGMENTS
            (begin = FL_CUSPS.cusp);
         Previous_Time = FL_CUSPS.time;
         First_Cusp = "false";
      ELSE
         UPDATE IN SEGMENTS
            (end = FL_CUSPS.cusp)
            WHERE SEGMENTS.begin_t EQ Previous_Time;
         IF FL_CUSPS.time NE MAX (FL_CUSPS.time)
         THEN
            INSERT INTO SEGMENTS
               (begin = FL_CUSPS.cusp);
            Previous_Time = FL_CUSPS.time;
END Cusps_To_Segments;
```

FIGURE 4-4
CUSPS_TO_SEGMENTS

The Grid_Chain_Generation (Figure 4-5) represents Airspace Probe's capability to cross-reference the aircraft's trajectory to the grid structure. The output of this routine is the list of all the volumes associated with the cells that the aircraft's trajectory intersects (in the horizontal plane).

In the element Set_Up_Segment_Scan (Figure 4-6), the slope for a trajectory segment is computed to determine the coordinate with the fastest change per unit distance. This is done so that the algorithm may increment the faster-changing variable (called the "independent variable") to step to the next row (or column) of grid cells assuming that the other coordinate will change at most one cell in either a positive or negative direction (see Figure 4-7). The element also identifies the cells containing the first and last points on the segment.

At each grid cell, the independent variable is incremented one step in grid-cell coordinates and the dependent variable is recalculated by the element Scan_Segment_To_Pick_Up_Cells (Figure 4-8). The next grid cell is determined from these new grid cell values. If it is found that the dependent variable has changed indicating a new row (or column) for the next grid cell, the element Add_Box (Figure 4-9) is invoked to find the intermediate cell which has been crossed (see Figure 4-10).

The element Add_Box determines which intermediate cell the trajectory passes through as follows (see Figure 4-11):

1. First, it is determined in what relation the current cell stands to the previous cell (upper right, etc.)

2. Second, the point between the two cells is found.

3. Next, the current trajectory segment is compared to the point between the cells. This enables the algorithm to determine if the trajectory segment passes to the right or left of the point. This uniquely determines the cell that the trajectory must pass through in order to reach the current cell.

4. Lastly, this intermediate cell is added to the grid chain and falls in the proper order.

The service utilities Get_Lower_Left_Corner_Points (Figure 4-12) and Put_Box_In_Grid_Chain (Figure 4-13) perform data retrieval and depositing to support Add_Box. The former

```
ROUTINE Grid_Chain_Generation;
PARAMETERS
   GRID_CHAIN_VOLUMES OUT;
REFER TO GLOBAL
   ENVIRONMENTAL_CELL_CONTENTS IN;
REFER TO SHARED LOCAL
   SEGMENTS IN;
DEFINE TABLES
   GRID_CHAIN_CELLS          The cells the trajectory intersects
      cell_id                   The cell identifier
      first_cusp_time           The time of the first cusp before the
                                   cell
   GRID_CHAIN_VOLUMES        The volumes within the cells which
                                   intersect the trajectory
      volume_id                 The volume identifier
      first_cusp_time           The time of the first cusp before the
                                   cell
   TEMP                      A temporary table
      volume_id                 The volume identifier;
DEFINE VARIABLES
   Prev_Box                  The last cell looked at
   Box                       The current cell
   Last_Box                  The final cell of the trajectory segment
   Slope                     The Y vs X slope of the segment
   Step_X                    The independent variable increment
   Step_Y                    The independent variable increment
   Indep_Var                 The independent variable;
   ##
   REPEAT FOR EACH SEGMENTS RECORD;
      CALL Set_Up_Segment_Scan (SEGMENTS.pair IN, Box OUT,
         Last_Box OUT, Slope OUT, Step_X OUT, Step_Y OUT,
         Indep_Var OUT, GRID_CHAIN_CELLS OUT);
      CALL Scan_Segment_To_Pick_Up_Cells (SEGMENTS.pair IN,
         Box IN, Last_Box IN, Slope IN, Step_X IN, Step_Y IN,
         Indep_Var IN, GRID_CHAIN_CELLS INOUT);
   REPEAT FOR EACH GRID_CHAIN_CELLS RECORD;
      TEMP = SELECT FIELDS volume_id
         FROM ENVIRONMENTAL_CELL_CONTENTS
         WHERE ENVIRONMENTAL_CELL_CONTENTS.cell_id EQ
            GRID_CHAIN_CELLS.cell_id;
      REPEAT FOR EACH TEMP RECORD;
         INSERT INTO GRID_CHAIN_VOLUMES
            (volume_id = TEMP.volume_id, first_cusp_time =
             GRID_CHAIN_CELLS.first_cusp_time);
END Grid_Chain_Generation;
```

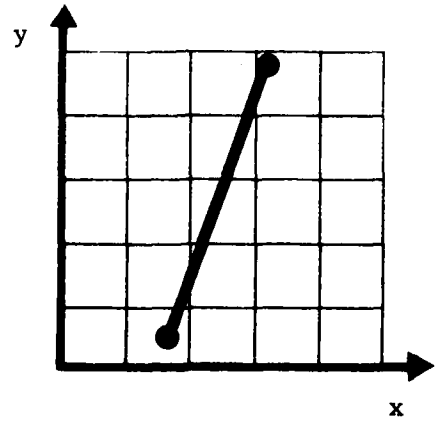
FIGURE 4-5
GRID_CHAIN_GENERATION

```
ROUTINE Set_Up_Segment_Scan;
PARAMETERS
   SEGMENT IN,
   Box OUT,
   Last_Box OUT,
   Slope OUT,
   Step_X OUT,
   Step_Y OUT,
   Indep_Var OUT
   GRID_CHAIN_CELLS OUT;
REFER TO GLOBAL
   Environmental_Cell_Width IN;
DEFINE TABLES
   SEGMENT                 The current trajectory segment
      begin_x                 The first cusp of the segment
      begin_y
      begin_z
      begin_t
      end_x                   The second cusp of the segment
      end_y
      end_z
      end_t
   GRID_CHAIN_CELLS        The cells intersecting the trajectory
      cell_id                 The cell identifier
      first_cusp_time         The time of the first cusp before the
                                 cell;
DEFINE VARIABLES
   Box                     The first cell intersected
   Last_Box                The last cell intersected
   Slope                   The slope of the segment with respect to
                              the independent variable
   Step_X                  The independent variable increment
   Step_Y                  The independent variable increment
   Indep_Var               The independent variable
   Delta_X                 The segment X extent
   Delta_Y                 The segment Y extent;
```

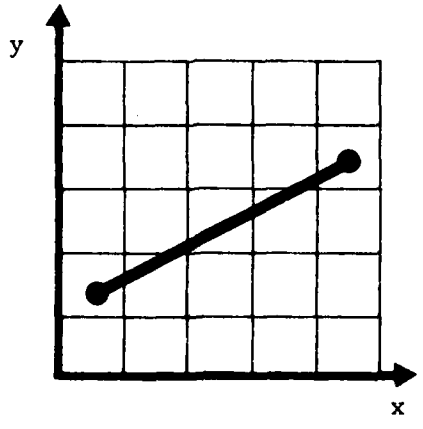
FIGURE 4-6
SET_UP_SEGMENT_SCAN

```
    CALL Grid (SEGMENT.begin_x IN, SEGMENT.begin_y IN,
       Box OUT);
    INSERT INTO GRID_CHAIN_CELLS
       (cell_id = Box, first_cusp_time = SEGMENT.begin_t);
    Delta_X = SEGMENT.end_x - SEGMENT.begin_x;
    Delta_Y = SEGMENT.end_y - SEGMENT.begin_y;
    Step_X = SIGN (Delta_X) * Environmental_Cell_Width;
    Step_Y = SIGN (Delta_Y) * Environmental_Cell_Width;
    Slope = Delta_Y/Delta_X;
    IF ABS(Slope) LT 1
    THEN
       Indep_Var = "X";
    ELSE
       Indep_Var = "Y";
       Slope = Delta_X/Delta_Y;
    CALL Grid (SEGMENT.end_x IN, SEGMENT.end_y IN,
       Last_Box OUT);
END Set_Up_Segment_Scan;
```

FIGURE 4-6 (Concluded)
SET_UP_SEGMENT_SCAN



(a) y is chosen as the independent variable.



(b) x is chosen as the independent variable

FIGURE 4-7
INDEPENDENT VARIABLE SELECTION

```
ROUTINE Scan_Segment_To_Pick_Up_Cells;
PARAMETERS
   SEGMENT IN,
   Box IN,
   Last_Box IN,
   Slope IN,
   Step_X IN,
   Step_Y IN,
   Indep_Var IN,
   GRID_CHAIN_CELLS INOUT;
REFER TO GLOBAL
   ENVIRONMENTAL_CELLS;
REFER TO SHARED LOCAL
   SEGMENTS IN;
DEFINE TABLES
   SEGMENT                    The current trajectory segment
      begin_x                 The first cusp of the segment
      begin_y
      begin_z
      begin_t
      end_x                   The second cusp of the segment
      end_y
      end_z
      end_t
   GRID_CHAIN_CELLS        The cells intersecting the trajectory
      cell_id                 The cell identifier
      first_cusp_time         The time of the first cusp before the
                                 cell;
DEFINE VARIABLES
   Box                  The first cell intersected
   Last_Box             The last cell intersected
   Slope                The slope of the segment with respect to the
                           independent variable
   Step_X               The independent variable increment
   Step_Y               The independent variable increment
   Indep_Var            The independent variable
   Prv_Box              The previous cell intersected
   Prv_Box_X            The minimum X value of the previous cell
   Prv_Box_Y            The minimum Y value of the previous cell
   Box_X                The minimum X value of the current cell
   Box_Y                The minimum Y value of the current cell
   Step_Count           The number of independent variable steps
   X                    The X coordinate of the current step
   Y                    The Y coordinate of the current step;
```

FIGURE 4-8
SCAN_SEGMENT_TO_PICK_UP_CELLS

```
      Step_Count = 0;
      REPEAT WHILE Box NE Last_Box;
         Step_Count = Step_Count + 1;
         Prv_Box = Box;
         SELECT FIELDS min_x,min_y
            FROM ENVIRONMENTAL_CELLS
            INTO X,Y
            WHERE ENVIRONMENTAL_CELLS.cell_id EQ Prv_Box;
         IF Indep_Var EQ "X"
         THEN
            X = X + Step_X;
            Y = SEGMENT.begin_y + Slope * Step_Count;
            CALL Grid (X IN, Y IN, Box OUT);
            SELECT FIELDS min_y
               FROM ENVIRONMENTAL_CELLS
               INTO Prev_Box_Y
               WHERE ENVIRONMENTAL_CELLS.cell_id EQ Prv_Box;
            SELECT FIELDS min_y
               FROM ENVIRONMENTAL_CELLS
               INTO Box_Y
               WHERE ENVIRONMENTAL_CELLS.cell_id EQ Box;
            IF Box_Y NE Prv_Box_Y
            THEN
               CALL Add_Box (Prv_Box IN, Box IN, SEGMENT IN,
               GRID_CHAIN_CELLS INOUT);
         ELSE
            Y = Y + Step_Y;
            X = SEGMENT.begin_x + Slope * Step_Count;
            CALL Grid (X IN, Y IN, Box OUT);
            SELECT FIELDS min_x
               FROM ENVIRONMENTAL_CELLS
               INTO Prv_Box_X
               WHERE ENVIRONMENTAL_CELLS.cell_id EQ Prv_Box;
            SELECT FIELDS min_x
               FROM ENVIRONMENTAL_CELLS
               INTO Box_X
               WHERE ENVIRONMENTAL_CELLS.cell_id EQ Box;
            IF Box_X NE Prv_Box_X
            THEN
               CALL Add_Box (Prv_Box IN, Box IN, SEGMENT IN,
               GRID_CHAIN_CELLS INOUT);
         INSERT INTO GRID_CHAIN_CELLS
            (cell_id = Box, first_cusp_time = SEGMENT.begin_t);
   END Scan_Segment_To_Pick_Up_Cells;
```

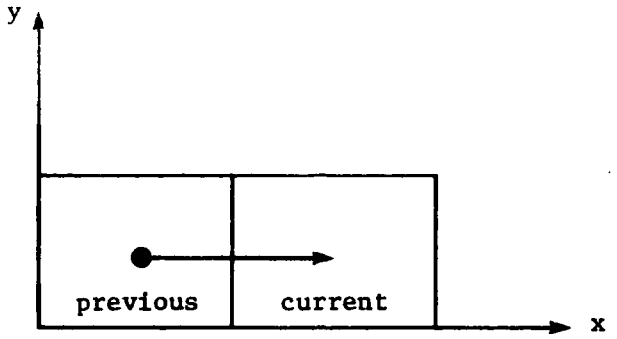
FIGURE 4-8 (Concluded)
SCAN_SEGMENT_TO_PICK_UP_CELLS

```
ROUTINE Add_Box;
PARAMETERS
   Prev_Box IN,
   Box IN,
   SEGMENT IN,
   GRID_CHAIN_CELLS INOUT;
DEFINE TABLES
   SEGMENT                  The current trajectory segment
      begin_x                  The first cusp of the segment
      begin_y
      begin_z
      begin_t
      end_x                    The second cusp of the segment
      end_y
      end_z
      end_t
   GRID_CHAIN_CELLS         The cells which interseot the trajectory
      cell_id                  The cell identifier
      first_cusp_time          The time of the first cusp before the
                                  cell;
DEFINE VARIABLES
   Prev_Box                 The previous cell intersected
   Box                      The current cell intersected
   Prev_Box_X               The minimum X value of the previous cell
   Prev_Box_Y               The minimum Y value of the previous cell
   Box_X                    The minimum X value of the current cell
   Box_Y                    The minimum Y value of the current cell
   Side                     The side of the line where the point is;
```

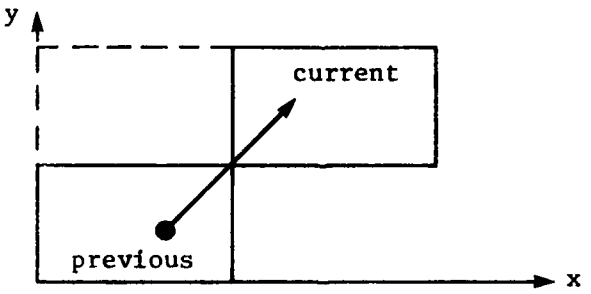
FIGURE 4-9
ADD_BOX

```
   CALL Get_Lower_Left_Corner_Points (Prev_Box IN, Box IN,
      Prev_Box_X OUT, Prev_Box_Y OUT, Box_X OUT, Box_Y OUT);
   CHOOSE CASE
      WHEN Box_X GT Prev_Box_X AND Box_Y GT Prev_Box_Y THEN
         CALL Linear_Discriminant_Classifier (SEGMENT.begin
            IN, SEGMENT.end IN, Box_X IN, Box_Y IN, Side OUT)
         IF Side EQ "left"
         THEN
            CALL Put_Box_In_Grid_Chain (Prev_Box_X IN,
               Box_Y IN, SEGMENT IN, GRID_CHAIN_CELLS INOUT);
         ELSE
            CALL Put_Box_In_Grid_Chain (Box_X IN, Prev_Box_Y IN,
               SEGMENT IN, GRID_CHAIN_CELLS INOUT);
      WHEN Box_X GT Prev_Box_X AND Box_Y LT Prev_Box_Y THEN
         CALL Linear_Discriminant_Classifier (SEGMENT.begin IN,
            SEGMENT.end IN, Box_X IN, Prev_Box_Y IN, side OUT)
         IF Side EQ "left"
         THEN
            CALL Put_Box_In_Grid_Chain (Box_X IN, Prev_Box_Y IN,
               SEGMENT IN, GRID_CHAIN_CELLS INOUT);
         ELSE
            CALL Put_Box_In_Grid_Chain (Prev_Box_X IN,
               Box_Y IN, SEGMENT IN, GRID_CHAIN_CELLS INOUT);
      WHEN Box_X LT Prev_Box_X AND Box_Y GT Prev_Box_Y THEN
         CALL Linear_Discriminant_Classifier (SEGMENT.begin IN,
            SEGMENT.end IN, Prev_Box_X IN, Box_Y IN, Side OUT);
         IF Side EQ "left"
         THEN
            CALL Put_Box_In_Grid_Chain (Box_X IN, Prev_Box_Y IN,
               SEGMENT IN, GRID_CHAIN_CELLS INOUT);
         ELSE
            CALL Put_Box_In_Grid_Chain (Prev_Box_X IN, Box_Y IN,
               SEGMENT IN, GRID_CHAIN_CELLS INOUT);
      WHEN Box_X LT Prev_Box_X AND Box_Y LT Prev_Box_Y THEN
         CALL Linear_Discriminant_Classifier (SEGMENT.begin IN,
            SEGMENT.end IN, Prev_Box_X IN, Prev_Box_Y IN, Side OUT)
         IF Side EQ "left"
         THEN
            CALL Put_Box_In_Grid_Chain (Prev_Box_X IN,
               Box_Y IN, SEGMENT IN, GRID_CHAIN_CELLS INOUT);
         ELSE
            CALL Put_Box_In_Grid_Chain (Box_X IN, Prev_Box_Y IN,
               SEGMENT IN, GRID_CHAIN_CELLS INOUT);
END Add_Box;
```

FIGURE 4-9 (Concluded)
ADD_BOX



(a) Independent variable is x and no change in y, therefore no intermediate grid cell.



(b) Independent variable is x and a change in y of +1 (grid cell coordinates) indicates an intermediate box is intersected (in dashed lines).

FIGURE 4-10
INTERMEDIATE GRID CELL RECOGNITION

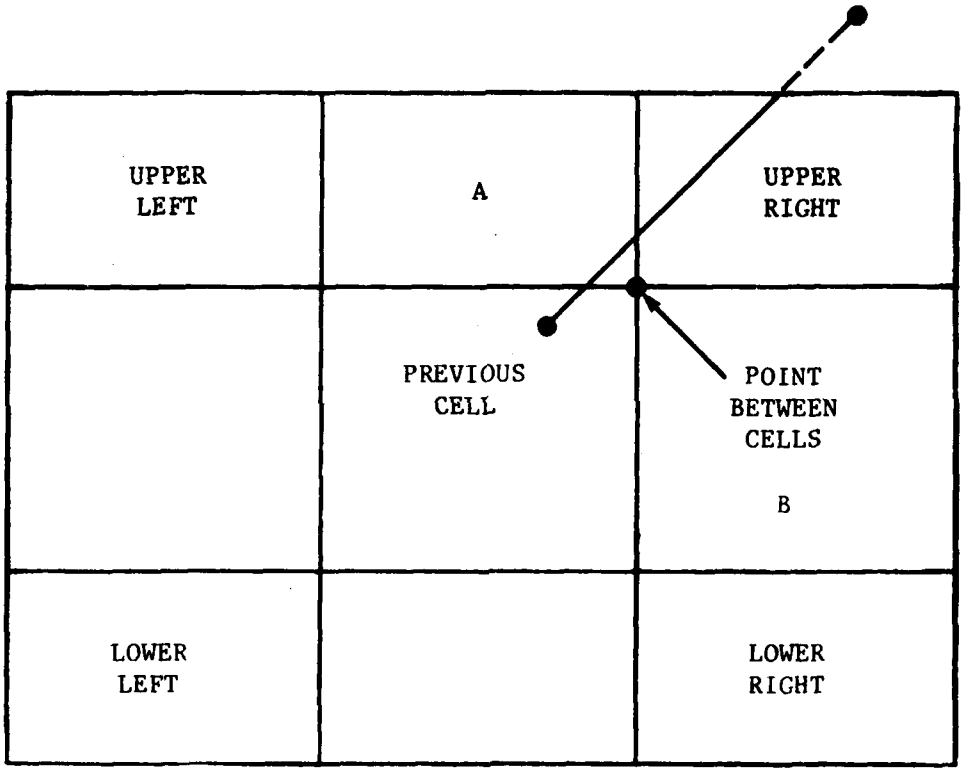


FIGURE 4-11
INTERMEDIATE GRID CELL DETERMINATION

```
ROUTINE Get_Lower_Left_Corner_Points;
PARAMETERS
   Prev_Box IN,
   Box IN,
   Prev_Box_X OUT,
   Prev_Box_Y OUT,
   Box_X OUT,
   Box_Y OUT;
REFER TO GLOBAL
   ENVIRONMENTAL_CELLS IN;
DEFINE VARIABLES
   Prev_Box              The previous cell intersected
   Box                   The current cell intersected
   Prev_Box_X            The minimum X value of the previous cell
   Prev_Box_Y            The minimum Y value of the previous cell
   Box_X                 The minimum X value of the current cell
   Box_Y                 The minimum Y value of the current cell;
##
   SELECT FIELDS min_x,min_y
      FROM ENVIRONMENTAL_CELLS
      INTO Prev_Box_X, Prev_Box_Y
      WHERE ENVIRONMENTAL_CELLS.cell_id EQ Prev_Box;
   SELECT FIELDS min_x,min_y
      FROM ENVIRONMENTAL_CELLS
      INTO Box_X, Box_Y
      WHERE ENVIRONMENTAL_CELLS.cell_id EQ Box;
END Get_Lower_Left_Corner_Points;
```

FIGURE 4-12
GET_LOWER_LEFT_CORNER_POINTS

```
ROUTINE Put_Box_In_Grid_Chain;
PARAMETERS
   X IN,
   Y IN,
   SEGMENT IN,
   GRID_CHAIN_CELLS INOUT;
DEFINE TABLES
   SEGMENT                The current trajectory segment
      begin_x                 The first cusp of the segment
      begin_y
      begin_z
      begin_t
      end_x                   The second cusp of the segment
      end_y
      end_z
      end_t
   GRID_CHAIN_CELLS       The cells intersecting the trajectory
      cell_id                 The cell identifier
      first_cusp_time         The time of the first cusp before the
                                 cell;
DEFINE VARIABLES
   X                      The X coordinate of the point
   Y                      The Y coordinate of the point
   Cell_Id                The cell which includes the point (X,Y);
##
   CALL Grid (X IN, Y IN, Cell_Id OUT);
   INSERT INTO GRID_CHAIN_CELLS
      (cell_id = Cell_Id, first_cusp_time = SEGMENT.begin_t);
END Put_Box_In_Grid_Chain;
```

FIGURE 4-13
PUT_BOX_IN_GRID_CHAIN

obtains the lower (least y) left (least x) hand corner point for two boxes. The latter inserts a cell identification (along with the time at the segment initial point) into the grid-chain table.

4.2 Second-Order Coarse Filter

4.2.1 Mission

First-Order Coarse Filter processing has identified a set of Nominee polygons. The Second-Order Coarse Filter is a finer filter which processes the First-Order Nominee polygons to reduce the set of potentially intersecting polygons. At this level of granularity, "close" is defined so as to include only those nominee polygons (approximated by the smallest right rectangle aligned square to the coordinate axes) which intersect the trajectory segments. The polygons passing this filter are examined in greater detail in the Fine Filter.

4.2.2 Design Considerations and Component Environment

In the Second-Order Coarse Filter, the algorithm accesses, for the first time in Airspace Probe, the actual dimensions of the four-dimensional polygons. However, the polygons, themselves, are not processed, but enclosed in a parallelepiped. The extents in the x, y, z, and t dimensions are used to construct the parallelepiped (Figure 4-14). One-dimensional intersection tests alone on this volume rapidly eliminate non-candidate polygons, especially those not intersecting the trajectory in the altitude and time dimensions (dimensions not incorporated into the First-Order Coarse Filter).

The sequence of elements associated with the Second-Order Coarse Filter is given in Figure 4-15. Program design language is provided in this section for each element of Figure 4-15 with the exception of Linear_Discriminant_Classifier. A description of that element is provided in Appendix B.

Input Data

The input data required by the Second-Order Coarse Filter consists of:

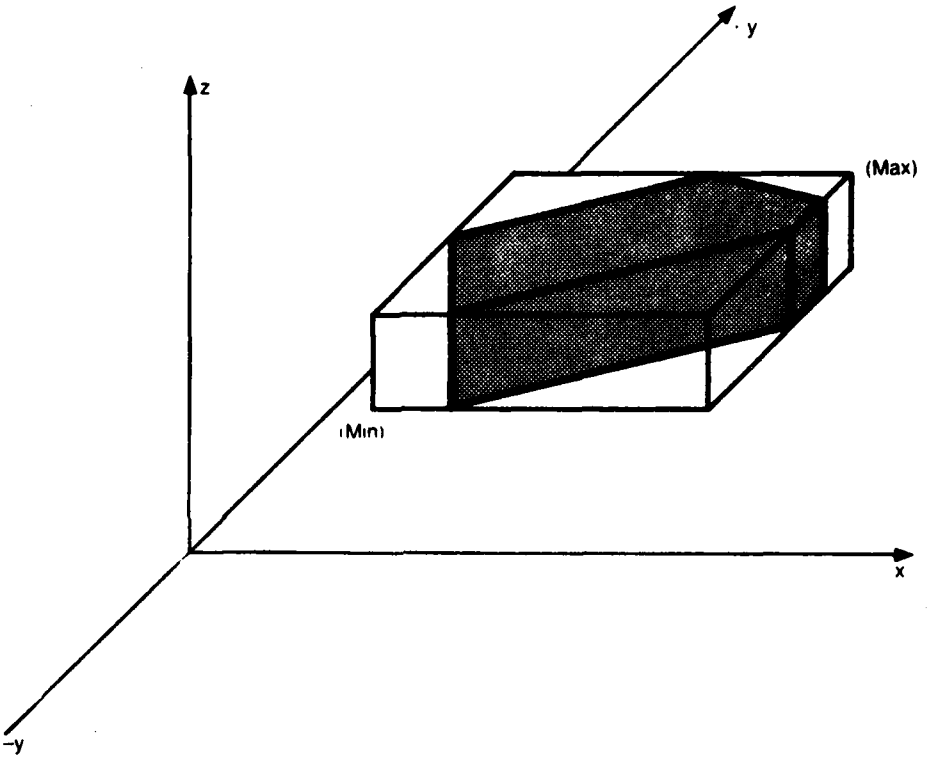


FIGURE 4-14
APPROXIMATION OF AN AIRSPACE BY
RECTANGLES IN EACH ORIENTATION PLANE

```
Second_Order_Coarse_Filter
   Retrieve_Polygon_Extents
   One_Dim_Checks
      Segment_Vs_Segment_Intersection
   Two_Dim_Checks
      Segment_Vs_Plane_Intersection
         Linear_Discriminant_Classifier
```

FIGURE 4-15
ELEMENTS OF THE SECOND-ORDER COARSE FILTER

- System Global Data Base

  - SPECIAL_USE_AIRSPACES

    The activation and deactivation times associated with individual polygons are retrieved to support time interval intersection tests.

  - VOLUME_COORDINATES

    The (x,y) coordinates of each vertex of each polygon are contained in this table. Only the maximum and minimum x's and y's are obtained. The ceiling and floor altitudes for the polygon are used to describe the vertical extent.

- Shared Local Data Base

  - SEGMENTS

    The aircraft's trajectory has been stored for Airspace Probe use as an ordered sequence of line segments. Each trajectory segment is checked for possible intersection with parallelepipeds containing First-Order Nominees.

  - FIRST_ORDER_NOMINEES

    This table contains the identity of each polygon thought to be "close" to the trajectory.

Output

The Second-Order Coarse Filter produces a list of nominee polygons which must be processed through the remainder of the Airspace Probe algorithm.

- Shared Local Data Base

  - SECOND_ORDER_NOMINEES

    The identities of the polygons which are identified by the Second-Order Coarse Filter are stored in this table. These polygons must have the property that a parallelepiped enclosing the volume intersects the trajectory of the aircraft.

4.2.3 Component Design Logic

The Second-Order Coarse Filter (Figure 4-16) examines each First-Order Nominee to determine if an intersection can exist with the aircraft trajectory. Each nominee is processed separately, first by obtaining the maximum and minimum x, y, z, and t values across the polygon. This first step is performed by the element Retrieve_Polygon_Extents (Figure 4-17).

Each trajectory segment, beginning with the cusp associated with the nominee, is examined for potential intersections. Each segment passing through the process undergoes tests against the rectilinear space circumscribed about the polygon being checked. To perform this test, a sequence of filtration steps are performed. The two major steps check if the aircraft trajectory segment intersects: (1) the extent of the polygon in single dimensions (Figure 4-18), and (2) the extent of the polygon in certain planes (Figure 4-19). If an intersection is not found at any particular step, the aircraft trajectory will not intersect the polygon. Consequently, the polygon is rejected as a Nominee immediately if this condition is detected.

The first step, given in the element One_Dim_Checks (Figure 4-20), sets up comparisons of the aircraft trajectory segment with the extent of the polygon. The comparisons done in Segment_Vs_Segment_Intersection (Figure 4-21) check to see if the 1-dimensional extent of the trajectory segment intersects the 1-dimensional extent of the polygon in corresponding dimensions. The order in which dimensions are checked should be ordered in such a way as to take advantage of the distribution of trajectory segment and polygon data. For example, if most aircraft trajectory segments input to the Second-Order Coarse Filter indicate that checking the altitude would drop half the cases but checking one of the horizontal dimensions would drop only a quarter of the cases, then the altitude check should be made before the horizontal checks.

The second step, given in the element Two_Dim_Checks (Figure 4-22), sets up comparisons of the extent of the aircraft trajectory to the extent of the polygon in various orientation planes. The comparisons done in Segment_Vs_Plane_Intersection (Figure 4-23) check to see if the 2-dimensional extent of the trajectory segment intersects the 2-dimensional extent of the polygon. Only the x-y, y-z, x-z, and z-t planes are examined. It is not necessary to check the x-t and y-t planes or the x-y-z volume since the planes checked account for these orientations. The Second-Order Coarse Filter examines the polygon from the various orientation planes in this coarse

```
ROUTINE Second_Order_Coarse_Filter;
REFER TO SHARED LOCAL
   SEGMENTS IN,
   FIRST_ORDER_NOMINEES IN,
   SECOND_ORDER_NOMINEES OUT;
DEFINE TABLES
   POLYGON_EXTENTS          The extents of the polygon in each
                               dimension
      min_x                    The minimum value of the x dimension
      min_y                    The minimum value of the y dimension
      min_z                    The minimum value of the z dimension
      min_t                    The minimum value of the t dimension
      max_x                    The maximum value of the x dimension
      max_y                    The maximum value of the y dimension
      max_z                    The maximum value of the z dimension
      max_t                    The maximum value of the t dimension;
DEFINE VARIABLES
   Segment_Intersection     This flags a segment intersection
   Plane_Intersection       This flags a plane intersection;
   ##
   REPEAT FOR EACH FIRST_ORDER_NOMINEES RECORD;
      CALL Retrieve_Polygon_Extents
        (FIRST_ORDER_NOMINEES.volume_id IN, POLYGON_EXTENTS OUT);
      REPEAT FOR EACH SEGMENTS RECORD
         WHERE SEGMENTS.begin_time GE
            FIRST_ORDER_NOMINEES.first_cusp_time AND
            FIRST_ORDER_NOMINEES.volume_id IS NOT IN
            SECOND_ORDER_NOMINEE.volume_id;
         CALL One_Dim_Checks (SEGMENTS.pair IN, POLYGON_EXTENTS IN,
                              Segment_Intersection OUT);
         IF Segment_Intersection EQ "true"
         THEN
            CALL Two_Dim_Checks (SEGMENTS.pair IN,
               POLYGON_EXTENTS IN, Plane_Intersection OUT);
            IF Plane_Intersection EQ "true"
            THEN
               INSERT INTO SECOND_ORDER_NOMINEES
                  (all = FIRST_ORDER_NOMINEES.all);
END Second_Order_Coarse_Filter;
```

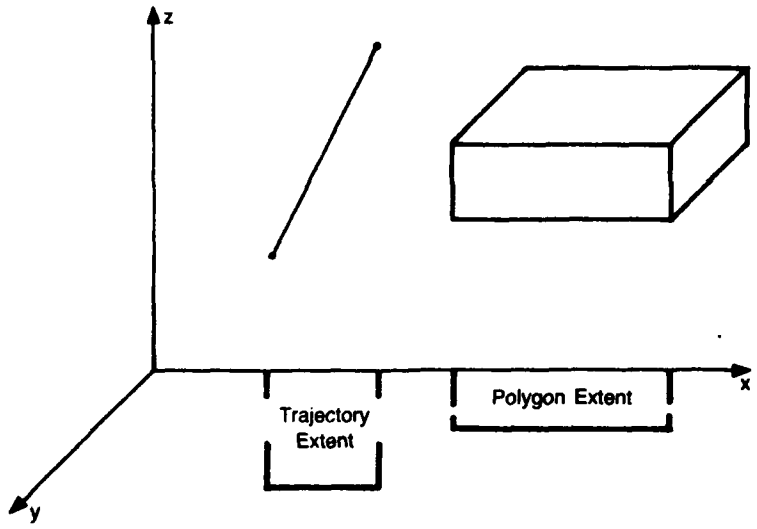
FIGURE 4-16
SECOND_ORDER_COARSE_FILTER

```
ROUTINE Retrieve_Polygon_Extents;
PARAMETERS
   Volume_Id IN,
   POLGON_EXTENTS OUT;
REFER TO GLOBAL
   SPECIAL_USE_AIRSPACES IN,
   VOLUME_COORDINATES IN;
DEFINE TABLES
   POLYGON_EXTENTS          The extents of the polygon in each
                               dimension
      min_x                    The minimum value of the X extent
      min_y                    The minimum value of the Y extent
      min_z                    The minimum value of the Z extent
      min_t                    The minimum value of the T extent
      max_x                    The maximum value of the X extent
      max_y                    The maximum value of the Y extent
      max_z                    The maximum value of the Z extent
      max_t                    The maximum value of the T extent;
DEFINE VARIABLES
   Volume_Id                The volume identifier
   Start_Time               The activation time of the polygon
   Stop_Time                The deactivation time of the polygon;
DEFINE CONSTANTS
   Earliest_Possible_Time      The earlist representable time
   Latest_Possible_Time        The latest representable time;
```

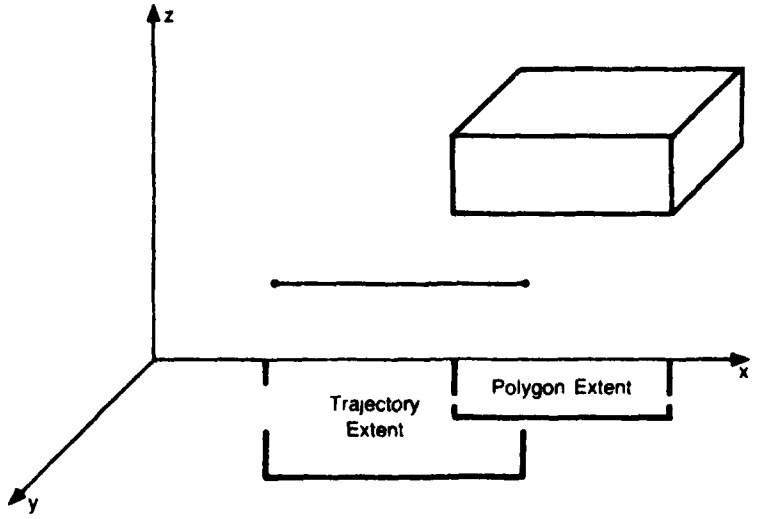
FIGURE 4-17
RETRIEVE_POLYGON_EXTENTS

```
    IF Volume_Id IS_IN SPECIAL_USE_AIRSPACES.volume_id
    THEN
       SELECT FIELDS start_time,stop_time
          FROM SPECIAL_USE_AIRSPACES
          INTO Start_Time,Stop_Time
          WHERE SPECIAL_USE_AIRSPACES.volume_id EQ Volume_Id;
    ELSE  # Volume Id must be for an E-MSAW area #
       Start_Time = Earliest_Possible_Time;
       Stop_Time = Latest_Possible_Time;
    INSERT INTO POLYGON_EXTENTS
       (min_t = Start_Time, max_t = Stop_Time);
    UPDATE IN POLYGON_EXTENTS
       (min_x = VOLUME_COORDINATES.x)
       WHERE VOLUME_COORDINATES.volume_id EQ Volume_Id AND
          VOLUME_COORDINATES.x EQ MIN (VOLUME_COORDINATES.x) AND
          POLYGON_EXTENTS.min_t EQ Start_Time;
    UPDATE IN POLYGON_EXTENTS
       (max_x = VOLUME_COORDINATES.x)
       WHERE VOLUME_COORDINATES.volume_id EQ Volume_Id AND
          VOLUME_COORDINATES.x EQ MAX (VOLUME_COORDINATES.x) AND
          POLYGON_EXTENTS.min_t EQ Start_Time;
    UPDATE IN POLYGON_EXTENTS
       (min_y = VOLUME_COORDINATES.y)
       WHERE VOLUME_COORDINATES.volume_id EQ Volume_Id AND
          VOLUME_COORDINATES.y EQ MIN (VOLUME_COORDINATES.y) AND
          POLYGON_EXTENTS.min_t EQ Start_Time;
    UPDATE IN POLYGON_EXTENTS
       (max_y = VOLUME_COORDINATES.y)
       WHERE VOLUME_COORDINATES.volume_id EQ Volume_Id AND
          VOLUME_COORDINATES.y EQ MAX (VOLUME_COORDINATES.y) AND
          POLYGON_EXTENTS.min_t EQ Start_Time;
    UPDATE IN POLYGON_EXTENTS
       (min_z = VOLUMES.floor_altitude)
       WHERE VOLUMES.volume_id EQ Volume_Id AND
          POLYGON_EXTENTS.min_t EQ Start_Time;
    UPDATE IN POLYGON_EXTENTS
       (max_z = VOLUMES.ceiling_altitude)
       WHERE VOLUMES.volume_id EQ Volume_Id AND
          POLYGON_EXTENTS.min_t EQ Start_Time;
END Retrieve_Polygon_Extents;
```

FIGURE 4-17 (Concluded)
RETRIEVE_POLYGON_EXTENTS



(a) No Intersection in the X-Dimension



(b) Intersection in the X-Dimension

FIGURE 4-18
TRAJECTORY/POLYGON ONE-DIMENSIONAL INTERSECTION

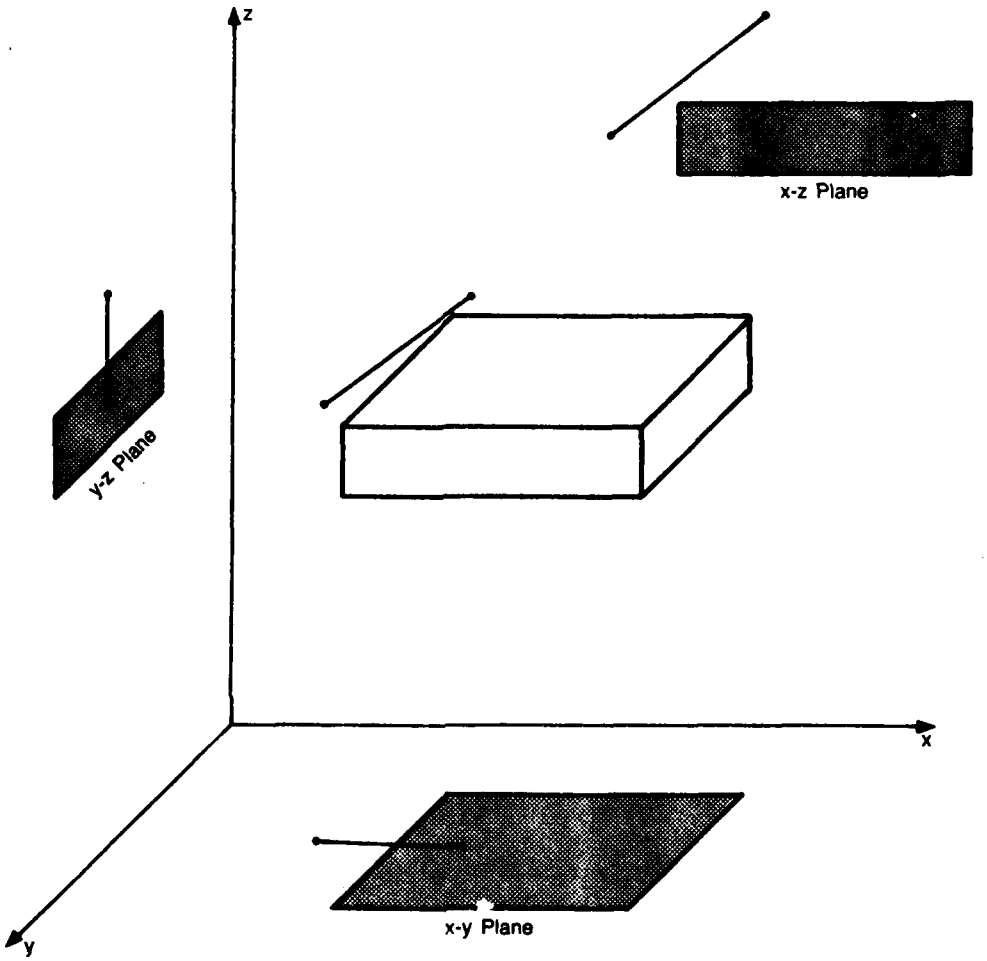


FIGURE 4-19
TRAJECTORY/POLYGON TWO-DIMENSIONAL INTERSECTION

```
ROUTINE One_Dim_Checks;
PARAMETERS
   SEGMENT IN,
   POLYGON_EXTENT IN,
   Segment_Intersection_In_All_Dimensions OUT;
DEFINE TABLES
   SEGMENT              The current trajectory segment
      begin_x              The first cusp of the segment
      begin_y
      begin_z
      begin_t
      end_x                The second cusp of the segment
      end_y
      end_z
      end_t
   POLYGON_EXTENT       The extent of the polygon in each
                           dimension
      min_x                The minimum value of the X extent
      min_y                The minimum value of the Y extent
      min_z                The minimum value of the Z extent
      min_t                The minimum value of the T extent
      max_x                The maximum value of the X extent
      max_y                The maximum value of the Y extent
      max_z                The maximum value of the Z extent
      max_t                The maximum value of the T extent;
DEFINE VARIABLES
   Segment_Intersection_In_All_Dimensions   Flag
   Segment_Intersection                     Flag;
```

FIGURE 4-20
ONE_DIM_CHECKS

```
    Segment_Intersection_In_All_Dimensions = "false";
    CALL Segment_Vs_Segment_Intersection (SEGMENT.begin_t IN,
       SEGMENT.end_t IN, POLYGON_EXTENT.min_t IN,
       POLYGON_EXTENT.max_t IN, Segment_Intersection OUT);
    IF Segment_Intersection EQ "true"
    THEN
       CALL Segment_Vs_Segment_Intersection (SEGMENT.begin_z IN,
          SEGMENT.end_z IN, POLYGON_EXTENT.min_z IN,
          POLYGON_EXTENT.max_z IN, Segment_Intersection OUT);
       IF Segment_Intersection EQ "true"
       THEN
          CALL Segment_Vs_Segment_Intersection (SEGMENT.begin_x IN,
             SEGMENT.end_x IN, POLYGON_EXTENT.min_x IN,
             POLYGON_EXTENT.max_x IN, Segment_Intersection OUT);
          IF Segment_Intersection EQ "true"
          THEN
             CALL Segment_Vs_Segment_Intersection
                (SEGMENT.begin_y IN, SEGMENT.end_y IN,
                POLYGON_EXTENT.min_y IN, POLYGON_EXTENT.max_y IN,
             Segment_Intersection OUT);
             IF Segment_Intersection EQ "true"
             THEN
                Segment_Intersection_In_All_Dimensions = "true";
END One_Dim_Checks;
```

FIGURE 4-20 (Concluded)
ONE_DIM_CHECKS

```
ROUTINE Segment_Vs_Segment_Intersection;
PARAMETERS
   Segment_Minimum IN,
   Segment_Maximum IN,
   Polygon_Minimum IN,
   Polygon_Maximum IN,
   Segment_Intersection OUT;
DEFINE VARIABLES
Segment_Minimum          The minimum value of the segment extent
                            for a given dimension
Segment_Maximum          The maximum value of the segment extent
                            for a given dimension
Polygon_Minimum          The minimum value of the polygon extent
                            for a given dimension
Polygon_Maximum          The maximum value of the polygon extent
                            for a given dimension
Segment_Intersection     The flag for a segment/polygon intersection;
##
   IF Segment_Minimum GT Polygon_Maximum OR
      Segment_Maximum LT Polygon_Minimum
   THEN
      Segment_Intersection = "false";
   ELSE
      Segment_Intersection = "true";
END Segment_Vs_Segment_Intersection;
```

FIGURE 4-21
SEGMENT_VS_SEGMENT_INTERSECTION

```
ROUTINE Two_Dim_Checks;
PARAMETERS
   SEGMENT IN,
   POLYGON_EXTENT IN,
   Plane_Intersection_In_All_Orientations OUT;
DEFINE TABLES
   SEGMENT              The current trajectory segment
      begin_x              The first cusp of the segment
      begin_y
      begin_z
      begin_t
      end_x                The second cusp of the segment
      end_y
      end_z
      end_t
   POLYGON_EXTENT       The extent of the polygon in each
                           dimension
      min_x                The minimum value of the X extent
      min_y                The minimum value of the Y extent
      min_z                The minimum value of the Z extent
      min_t                The minimum value of the T extent
      max_x                The maximum value of the X extent
      max_y                The maximum value of the Y extent
      max_z                The maximum value of the Z extent
      max_t                The maximum value of the T extent;
DEFINE VARIABLES
   Plane_Intersection_In_All_Orientations  Flag
   Plane_Intersection                      Flag;
```

FIGURE 4-22
TWO_DIM_CHECKS

```
      Plane_Intersection_In_All_Orientations = "false";
      CALL Segment_Vs_Plane_Intersection (SEGMENT.begin_t IN,
         SEGMENT.begin_z IN, SEGMENT.end_t IN, SEGMENT.end_z IN,
         POLYGON_EXTENT.min_t IN, POLYGON_EXTENT.max_t IN,
         POLYGON_EXTENT.min_z IN, POLYGON_EXTENT.max_z IN,
         Plane_Intersection OUT);
      IF Plane_Intersection EQ "true"
      THEN
         CALL Segment_Vs_Plane_Intersection (SEGMENT.begin_x IN,
            SEGMENT.begin_z IN, SEGMENT.end_x IN, SEGMENT.end_z IN,
            POLYGON_EXTENT.min_x IN, POLYGON_EXTENT.max_x IN,
            POLYGON_EXTENT.min_z IN, POLYGON_EXTENT.max_z IN,
            Plane_Intersection OUT);
         IF Plane_Intersection EQ "true"
         THEN
            CALL Segment_Vs_Plane_Intersection (SEGMENT.begin_y IN,
               SEGMENT.begin_z IN, SEGMENT.end_y IN,
               SEGMENT.end_z IN, POLYGON_EXTENT.min_y IN,
               POLYGON_EXTENT.max_y IN, POLYGON_EXTENT.min_z IN,
               POLYGON_EXTENT.max_z IN, Plane_Intersection OUT);
            IF Plane_Intersection EQ "true"
            THEN
               CALL Segment_Vs_Plane_Intersection
                  (SEGMENT.begin_x IN, SEGMENT.begin_y IN,
                  SEGMENT.end_x IN, SEGMENT.end_y IN,
                  POLYGON_EXTENT.min_x IN, POLYGON_EXTENT.max_x IN,
                  POLYGON_EXTENT.min_y IN, POLYGON_EXTENT.max_y IN,
                  Plane_Intersection OUT);
               IF Plane_Intersection EQ "true"
               THEN
                  Plane_Intersection_In_All_Orientations = "true";
END Two_Dim_Checks;
```

FIGURE 4-22 (Concluded)
TWO_DIM_CHECKS

```
ROUTINE Segment_Vs_Plane_Intersection;
PARAMETERS
   Segment_Start_U IN,
   Segment_Start_V IN,
   Segment_End_U IN,
   Segment_End_V IN,
   Polygon_Minimum_U IN,
   Polygon_Minimum_V IN,
   Polygon_Maximum_U IN,
   Polygon_Maximum_V IN;
   Segment_Intersection OUT;
DEFINE VARIABLES
   Segment_Start_U          The value of the U coordinate for the
                               first cusp of the segment
   Segment_Start_V          The value of the V coordinate for the
                               first cusp of the segment
   Segment_End_U            The value of the U coordinate for the
                               second cusp of the segment
   Segment_End_V            The value of the V coordinate for the
                               second cusp of the segment
   Polygon_Minimum_U        The minimum U extent of the polygon
   Polygon_Minimum_V        The minimum V extent of the polygon
   Polygon_Maximum_U        The maximum U extent of the polygon
   Polygon_Maximum_V        The maximum V extent of the polygon
   Plane_Intersection       The segment/plane intersection flag
   First_Side               The side of the segment on which the
                               first polygon vertex lies
   Side                     The side of the segment on which the
                            current polygon vertex lies;
```

FIGURE 4-23
SEGMENT_VS_PLANE_INTERSECTION

```
    Plane_Intersection = "true";
    CALL Linear_Discriminant_Classifier (Segment_Start_U IN,
       Segment_Start_V IN, Segment_End_U IN, Segment_End_V IN,
       Polygon_Minimum_U IN, Polygon_Minimum_V IN, First_Side OUT);
    CALL Linear_Discriminant_Classifier (Segment_Start_U IN,
       Segment_Start_V IN, Segment_End_U IN, Segment_End_V IN,
       Polygon_Maximum_U IN, Polygon_Minimum_V IN, Side OUT);
    IF Side EQ First_Side
    THEN
       CALL Linear_Discriminant_Classifier (Segment_Start_U IN,
          Segment_Start_V IN, Segment_End_U IN, Segment_End_V IN,
          Polygon_Maximum_U IN, Polygon_Maximum_V IN, Side OUT);
       IF Side EQ First_Side
       THEN
          CALL Linear_Discriminant_Classifier (Segment_Start_U IN,
             Segment_Start_V IN, Segment_End_U IN, Segment_End_V IN,
             Polygon_Minimum_U IN, Polygon_Maximum_V IN, Side OUT);
          IF Side EQ First_Side
          THEN
             Plane_Intersection = "false";
END Segment_Vs_Plane_Intersection;
```

FIGURE 4-23 (Concluded)
SEGMENT_VS_PLANE_INTERSECTION

manner. The entire polygon is not examined but rather the smallest rectilinear space square to the coordinate axes circumscribed about it.

To determine whether a segment intersects a given rectangle coarsely describing the extent of the polygon in a certain orientation plane, a linear discriminant is used. The Linear_Discriminant_Classifier is described in Appendix B. With the information it provides , one can classify points in the orientation plane as being left or right of the trajectory segment. A trajectory segment will intersect the rectangle about the polygon extent (in a given plane) if points of the polygon are found both to the left and to right of the segment (i.e. a line of the rectangle must cross the segment).

4.3 Fine Filter Processing

4.3.1 Mission

The Second-Order Coarse Filter processing has identified a set of Nominee polygons that are close to, but do not necessarily intersect, the aircraft's trajectory. The Fine Filter processing now must determine whether the given polygons do indeed intersect the aircraft's trajectory. The processing for each polygon is more involved than that in the coarse filters since the polygons may have concave sides and the exact points of intersection in 4-space must be determined. The information found by the Fine Filter is passed on to Encounter Processing to set up the relevant global data structures.

4.3.2 Design Considerations and Component Environment

In the Fine Filter, the coordinates of the vertex points of each Second-Order Nominee are used to construct line segments to test for intersection with a trajectory segment. For efficiency reasons, the logic should consider convex and nonconvex polygons separately.

The sequencing of elements associated with the Fine Filter is given in Figure 4-24. Program design language is provided in this section for each element of Figure 4-24 with the exception of Find_Polygon_Boundary_Intersections, Linear_Discriminant_Classifier, and Time_To_Violation. A description of these elements is provided in Appendix B.

```
Fine_Filter
   Convex_Polygon_Intersection_Check
      Find_Polygon_Boundary_Intersections
         Linear_Discriminant_Classifier
         Time_To_Violation
   Mixed_Polygon_Intersection_Check
      Find_Polygon_Boundary_Intersections
         Linear_Discriminant_Classifier
         Time_To_Violation
   Group_Into_Intersection_Pairs
   Vertical_Violation_Check
   Find_Exact_Violation_Points
```

FIGURE 4-24
ELEMENTS OF THE FINE FILTER

Input

The input data required by the Fine Filter consists of:

- System Global Data Base
  - VOLUMES

    The volume type is obtained--either "convex" or "mixed." This field is used to determine which polygon intersection test routine to use.

  - VOLUME_COORDINATES

    The vertex points of the polygons are obtained for line intersection tests.

- Shared Local Data Base
  - SEGMENTS

    The trajectory of the aircraft is stored locally as an ordered sequence of line segments.

  - SECOND_ORDER_NOMINEES

    This table contains the identity of each polygon passing the tests of the Second-Order Coarse Filter.

Output

The Fine Filter produces a list of environmental conflicts which are stored locally.

- ENVIRONMENTAL_CONFLICT_DATA

  This table contains all information necessary to identify an encounter. This table is input to Encounter Processing.

4.3.3 Component Design Logic

The Fine Filter (Figure 4-25) examines each Second-Order Nominee separately to determine if an intersection truly exists with the aircraft trajectory. To perform this task, three steps are taken. First, the extent of the horizontal penetration is determined. Second, the extent of the vertical penetration is determined. And third, the points of intersection are found.

```
ROUTINE Fine_Filter;
REFER TO GLOBAL
   VOLUMES IN;
REFER TO SHARED LOCAL
   SEGMENTS IN
   SECOND_ORDER_NOMINEES IN;
DEFINE TABLES
   SEGMENT_INTERSECTION_POINTS     The table of all intersections
      time                 The time of the intersection
      type                 Notes a boundary or interior intersection
      last_cusp_time       The time of the last cusp before the
                              intersection
   INTERSECTION_PAIRS      The table of all in/out intersections
      start_time              The time of the intersection going in
      stop_time               The time of the intersection going out
      begin_x                 Start cusp of segment on which intersection
      begin_y                    occurred
      begin_z
      begin_t
      end_x                   End cusp of segment on which intersection
      end_y                      occurred
      end_z
      end_t
      all AGGREGATE (start_time,stop_time,begin_x,begin_y,begin_z,
                     begin_t,end_x,end_y,end_z,end_t);
DEFINE VARIABLES
   Polygon_Type            Concave or mixed concave/convex polygon
   Vertical_Violation      Flag indicating intersection in the
                           vertical dimension
   Encounter               Flag indicating that the trajectory
                           intersects the polygon;
```

FIGURE 4-25
FINE_FILTER

```
    REPEAT FOR EACH SECOND_ORDER_NOMINEE RECORD;
       SELECT FIELDS polygon_type
          FROM VOLUMES
          INTO Polygon_Type
          WHERE VOLUMES.volume_id EQ SECOND_ORDER_NOMINEE.volume_id;
       REPEAT FOR EACH SEGMENTS RECORD
          WHERE SEGMENTS.begin_time GE
             SECOND_ORDER_NOMINEE.first_cusp_time;
          IF Polygon_Type EQ "convex"
          THEN
             CALL Convex_Polygon_Intersection_Check
                (SEGMENTS.pair IN, SECOND_ORDER_NOMINEES.volume_id IN,
                 SEGMENT_INTERSECTION_POINTS INOUT);
          ELSE
             CALL Mixed_Polygon_Intersection_Check
                (SEGMENTS.pair IN,
                 SECOND_ORDER_NOMINEES.volume_id IN,
                 SEGMENT_INTERSECTION_POINTS INOUT);
       CALL Group_Into_Intersection_Pairs
            (SEGMENT_INTERSECTION_POINTS IN, INTERSECTION_PAIRS OUT);
       Encounter = "false";
       REPEAT FOR EACH INTERSECTION_PAIRS RECORD;
          CALL Vertical_Violation_Check (INTERSECTION_PAIRS INOUT,
            SECOND_ORDER_NOMINEES.volume_id IN,
            Vertical_Violation OUT);
          IF Vertical_Violation EQ "true";
          THEN
             Encounter = "true";
          ELSE
             DELETE FROM INTERSECTION_PAIRS
                WHERE (INTERSECTION_PAIRS.all EQ
                   INTERSECTION_PAIRS.all);
       IF Encounter EQ "true";
       THEN
          CALL Find_Exact_Violation_Points (INTERSECTION_PAIRS IN,
             SECOND_ORDER_NOMINEES.volume_id IN);
END Fine_Filter;
```

FIGURE 4-25 (Concluded)
FINE_FILTER

The first step deals with the horizontal extent only. Convex and Mixed polygons are treated differently in Convex_Polygon_Intersection_Check (Figure 4-26) and a Mixed_Polygon_Intersection_Check (Figure 4-27), respectively; the objective is to check all sides of the polygon for possible penetrations. This is done to screen out polygons which are very close to the aircraft's trajectory but do not intersect it. The details of this step are taken primarily from E-MSAW documentation [8,9]. Appendix C of this document contains updated details. The outcome of this process is information concerning the preliminary points of penetration (more specifically, the times associated with these points) in the horizontal extent. The number of intersection points may be one (for trajectories which begin or end in the polygon), two (for Convex and Mixed polygons), or more than two (for Mixed polygons).

The horizontal points of penetration to be considered are selected by considering the relation of the trajectory segment to the polygon. If the trajectory begins/ends in the polygon, then the starting/stopping point is considered as one point of an intersection pair with the intersection of the polygon side as the other. If the trajectory segment intersects only two sides of either type polygon, these are used. If the trajectory intersects a Mixed polygon in several places, a set of intersection pairs will be formed. Each pair will consist of an entry point and exit point from the polygon. These penetration points are grouped into "enter-exit" pairs in the element Group_Into_Intersection_Pairs (Figure 4-28).

The second step examines the vertical extent of penetration for each intersection pair. This is done in the element Vertical_Violation_Check (Figure 4-29). Figures 4-30 and 4-31 illustrate this step. Since the polygons are defined by minimum and maximum altitudes, there can be at most two points of penetration per segment in vertical extent. The process begins by assuming the vertical points lie immediately over the horizontal points of penetration (h_1 and h_2) and intersect the aircraft trajectory (denoted by "x"s). Then these points are compared with the extent of the polygon in the vertical dimension. If both vertical penetration points lie within this range, the exact points of intersection have already been found and the polygon is added to the list of encounters. If both of the vertical penetration points lie above the polygon or both lie below, then the trajectory does not intersect the polygon, the polygon is screened out and rejected.

```
ROUTINE Convex_Polygon_Intersection_Check;
PARAMETERS
   SEGMENT IN,
   Volume_Id IN,
   SEGMENT_INTERSECTION_POINTS INOUT;
DEFINE TABLES
   SEGMENT              The current trajectory segment
      x1                   The first cusp point of the segment
      y1
      z1
      t1
      x2                   The second cusp point of the segment
      y2
      z2
      t2
      begin AGGREGATE (x1,y1)
      end AGGREGATE (x2,y2)
   SEGMENT_INTERSECTION_POINTS   The intersections with the polygon
      time                          The intersection time
      type                          The intersection location
                                       "boundary"  of "interior"
      last_cusp_time                The time of the last cusp before
                                       the intersection
   ORIENTATIONS            The orientation of the cusps (IN or OUT)
      begin_orient            The orientation of the first cusp
      begin_time              The time of the first cusp
      end_orient              The orientation of the end cusp
      end_time                The time of the end cusp
      time                    The time to violation;
DEFINE VARIABLES
   Iosum_Counter           The IN/OUT intersection counter
   Begin_Orient            The orientation of the first cusp
   Begin_Time              The time of the first cusp
   End_Orient              The orientation of the second cusp
   End_Time                The time of the second cusp;
```

FIGURE 4-26
CONVEX_POLYGON_INTERSECTION_CHECK

```
CALL Find_Polygon_Boundary_Intersections (SEGMENT IN, Volume_Id
   IN, ORIENTATIONS OUT, SEGMENT_INTERSECTION_POINTS INOUT);
CHOOSE CASE
   WHEN Iosum_Counter EQ 2 THEN
      INSERT INTO SEGMENT_INTERSECTION_POINTS
         (time = SEGMENT.t1, type = "interior",
         last_cusp_time = SEGMENT.t1);
      INSERT INTO SEGMENT_INTERSECTION_POINTS
         (time = SEGMENT.t2, type = "interior",
         last_cusp_time = SEGMENT.t2);
   WHEN Iosum_Counter EQ -2 THEN
      # do nothing #;
   OTHERWISE
      SELECT FIELDS begin_orient, begin_time
         FROM ORIENTATIONS
         INTO Begin_Orient, Begin_Time
         WHERE ORIENTATIONS.time EQ MIN (ORIENTATIONS.time)
      IF Begin_Orient EQ "in"
      THEN
         INSERT INTO SEGMENT_INTERSECTION_POINTS
            (time = Begin_Time, type = "interior",
             last_cusp_time = Begin_Time);
      SELECT FIELDS end_orient, end_time
         FROM ORIENTATIONS
         INTO End_Orient, End_Time
         WHERE ORIENTATIONS.time EQ MAX (ORIENTATIONS.time)
      IF End_Orient EQ "in"
      THEN
         INSERT INTO SEGMENT_INTERSECTION_POINTS
            (time = End_Time, type = "interior",
             last_cusp_time = End_Time);
END Convex_Polygon_Intersection_Check;
```

FIGURE 4-26 (Concluded)
CONVEX_POLYGON_INTERSECTION_CHECK

```
ROUTINE Mixed_Polygon_Intersection_Check;
PARAMETERS
   SEGMENT IN,
   Volume_Id IN,
   SEGMENT_INTERSECTION_POINTS INOUT;
DEFINE TABLES
   SEGMENT              The current trajectory segment
      x1                   The first cusp point of the segment
      y1
      z1
      t1
      x2                   The second cusp point of the segment
      y2
      z2
      t2
      begin AGGREGATE (x1,y1)
      end AGGREGATE (x2,y2)
   SEGMENT_INTERSECTION_POINTS   The intersections with the polygon
      time                          The intersection time
      type                          The intersection location
                                       "boundary" of "interior"
      last_cusp_time                The time of the last cusp before
                                       the intersection
   ORIENTATIONS            The orientation of the cusps (IN or OUT)
      begin_orient            The orientation of the first cusp
      begin_time              The time of the first cusp
      end_orient              The orientatin of the end cusp
      end_time                The time of the end cusp
      time                    The time to violation;
DEFINE VARIABLES
   Iosum_Counter           The IN/OUT intersection counter
   Begin_Orient            The orientation of the first cusp
   Begin_Time              The time of the first cusp
   End_Orient              The orientation of the second cusp
   End_Time                The time of the second cusp;
```

FIGURE 4-27
MIXED_POLYGON_INTERSECTION_CHECK

```
    CALL Find_Polygon_Boundary_Intersections (SEGMENT IN, Volume_Id
       IN, ORIENTATIONS OUT, SEGMENT_INTERSECTION_POINTS INOUT);
    CHOOSE CASE
       WHEN Iosum_Counter EQ 2 THEN
          INSERT INTO SEGMENT_INTERSECTION_POINTS
             (time = SEGMENT.t1, type = "interior",
              last_cusp_time = SEGMENT.t1);
       WHEN Iosum_Counter EQ -2 THEN
          # do nothing #;
       OTHERWISE
          SELECT FIELDS begin_orient, begin_time
             FROM ORIENTATIONS
             INTO Begin_Orient, Begin_Time
             WHERE ORIENTATIONS.time EQ MIN (ORIENTATIONS.time);
          IF Begin_Orient EQ "in"
          THEN
             INSERT INTO SEGMENT_INTERSECTION_POINTS
                (time = Begin_Time, type = "interior",
                 last_cusp_time = Begin_Time);
          SELECT FIELDS end_orient, end_time
             FROM ORIENTATIONS
             INTO End_Orient, End_Time
             WHERE ORIENTATIONS.time EQ MAX (ORIENTATIONS.time);
          IF End_Orient EQ "in"
          THEN
             INSERT INTO SEGMENT_INTERSECTION_POINTS
                (time = End_Time, type = "interior",
                last_cusp_time = End_Time);
END Mixed_Polygon_Intersection_Check;
```

FIGURE 4-27 (Concluded)
MIXED_POLYGON_INTERSECTION_CHECK

```
ROUTINE Group_Into_Intersection_Pairs;
PARAMETERS
   SEGMENT_INTERSECTION_POINTS IN,
   INTERSECTION_PAIRS OUT;
REFER TO SHARED LOCAL
   SEGMENTS IN;
DEFINE TABLES
   SEGMENT_INTERSECTION_POINTS  The intersection points
      time                         The intersection time
      type                         The location of the intersection
                                      (boundary or interior)
      last_cusp_time               The time associated with the last
                                      cusp before the intersection
   INTERSECTION_PAIRS      The intersections grouped into the
                              enter and exit violation points
      start_time              The time associated with the
                                 intersection entering the polygon
      stop_time               The time associated with the
                                 intersection exiting the polygon
      begin_x                 The segment on which the intersection
                                 lies
      begin_y                 (the first cusp)
      begin_z
      begin_t
      end_x                   (the second cusp)
      end_y
      end_z
      end_t
      segment AGGREGATE (begin_x,begin_y,begin_z,begin_t,
                         end_x,end_y,end_z,end_t);
DEFINE VARIABLES
   Point                   The flag indicating the first or last
                              point of a segment
   Start_Time              The time of the first cusp of a segment;
```

FIGURE 4-28
GROUP_INTO_INTERSECTION_PAIRS

```
    DELETE FROM SEGMENT_INTERSECTION_POINTS
       WHERE NOT ( SEGMENT_INTERSECTION_POINTS.time EQ
          MIN (SEGMENT_INTERSECTION_POINTS.time) OR
          SEGMENT_INTERSECTION_POINTS.type EQ "boundary" OR
          SEGMENT_INTERSECTION_POINTS.TIME EQ
          MAX (SEGMENT_INTERSECTION_POINTS.time) );
    Point = "start";
    REPEAT FOR EACH SEGMENT_INTERSECTION_POINTS RECORD
       IF Point EQ "start"
       THEN
          Start_Time = SEGMENT_INTERSECTION_POINTS.time;
          SEGMENT = SELECT FIELDS ALL
             FROM SEGMENTS
             WHERE (SEGMENTS.begin_t EQ Start_Time);
          INSERT INTO INTERSECTION_PAIRS (start_time =
             SEGMENT_INTERSECTION_POINTS.time, segment = SEGMENT);
          Point = "stop";
       ELSE
          UPDATE IN INTERSECTION_PAIRS (stop_time =
             SEGMENT_INTERSECTION_POINTS.time)
             WHERE INTERSECTION_PAIRS.start_time EQ Start_Time;
          Point = "start";
END Group_Into_Intersection_Pairs;
```

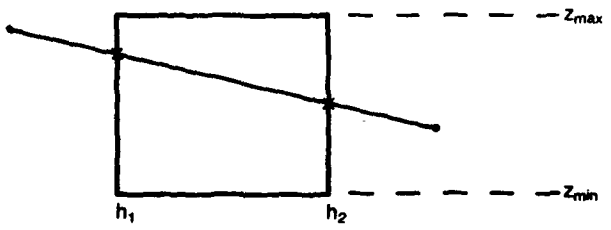
FIGURE 4-28 (Concluded)
GROUP_INTO_INTERSECTION_PAIRS

```
ROUTINE Vertical_Violation_Check;
PARAMETERS
   INTERSECTION_DATA INOUT,
   Volume_Id IN
   Vertical_Violation OUT;
REFER TO GLOBAL
   VOLUMES IN;
DEFINE TABLES
   INTERSECTION_DATA      The intersections grouped into the
                             enter and exit violation points
      start_time             The time associated with the
                                intersection entering the polygon
      stop_time              The time associated with the
                                intersection exiting the polygon
      begin_x                The segment on which the intersection
                                lies
      begin_y                (the first cusp)
      begin_z
      begin_t
      end_x                  (the second cusp)
      end_y
      end_z
      end_t;
DEFINE VARIABLES
   Volume_Id              The volume identifier
   Vertical_Violation     The flag indicating that a violation in
                             the vertical dimension has occurred
   Floor                  The minimum vertical extent of the polygon
   Ceiling                The maximum vertical extent of the polygon
   Start_Time             The time of entrance violation
   Stop_Time              The time of exit violation
   Begin_T                The first cusp time
   Begin_Z                The first cusp altitude
   End_T                  The second cusp time
   End_Z                  The second cusp altitude
   Z_Vel                  Average vertical velocity on the segment
   T1                     Intermediate variables
   T2
   Z1
   Z2;
```

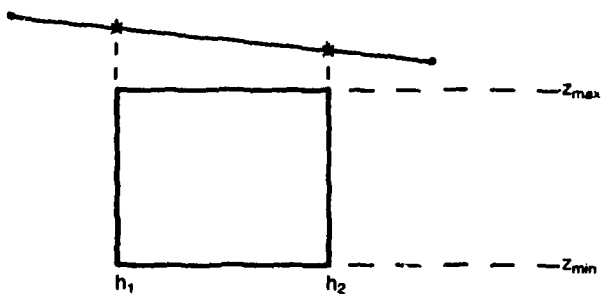
FIGURE 4-29
VERTICAL_VIOLATION_CHECK

```
    SELECT FIELDS floor_altitude,ceiling_altitude
       FROM VOLUMES
       INTO Floor,Ceiling
       WHERE VOLUMES.volume_id EQ Volume_Id;
    SELECT FIELDS start_time,stop_time,begin_z,begin_t,end_z,end_t
       FROM INTERSECTION_DATA
       INTO Start_Time,Stop_Time,Begin_Z,Begin_T,End_Z,End_T;
    Z_Vel = (Begin_Z - End_Z) / (End_T - Begin_T);
    T1 = Start_Time - Begin_T;
    T2 = Stop_Time - End_T;
    Z1 = Z_Vel * T1 + Begin_Z;
    Z2 = Z_Vel * T2 + Begin_Z;
    IF (Z1 GT Ceiling AND Z2 GT Ceiling) OR
       (Z1 LT Floor AND Z2 LT Floor)
    THEN
       Vertical_Violation = "false";
    ELSE
       IF Z1 GT Ceiling
       THEN
          Start_Time = T1 + (Ceiling - Z1)/Z_Vel;
       IF Z1 LT Floor
       THEN
          Start_Time = T1 + (Floor - Z1)/Z_Vel;
       IF Z2 GT Ceiling
       THEN
          Stop_Time = T2 + (Ceiling - Z2)/Z_Vel;
       IF Z2 LT Floor
       THEN
          Stop_Time = T2 + (Floor - Z2)/Z_Vel;
       UPDATE IN INTERSECTION_DATA
          (start_time = Start_Time, stop_time = Stop_Time);
       Vertical_Violation = "true";
END Vertical_Violation_Check;
```

FIGURE 4-29 (Concluded)
VERTICAL_VIOLATION_CHECK



(a) Exact Intersection Found



(b) No Intersection Found

FIGURE 4-30
VERTICAL PENETRATION CHECK

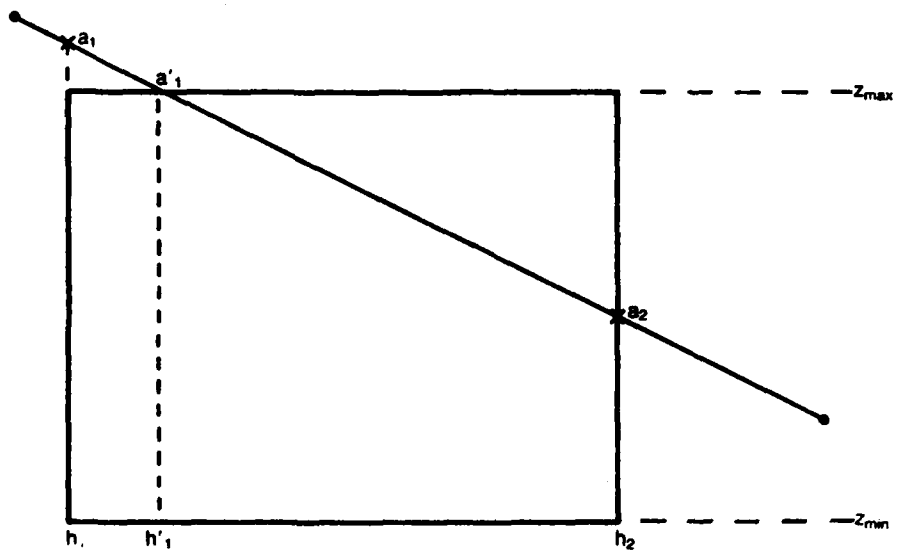


FIGURE 4-31
VERTICAL PENETRATION DETERMINATION

If neither of the above cases hold, then the process must determine the true points of intersection. This is done in Find_Exact_Violation_Points (Figure 4-32) by first computing the difference between the assumed penetration altitude (denoted by "a_1"s in Figure 4-31) and the altitude boundary of the polygon. From the aircraft's vertical velocity (approximated by consideration of segment data) and the above difference in altitude, a new time of intersection is computed.

After the intersection times have been found, the true four-dimensional extent of penetration is determined by projecting in x,y and z using their respective derived velocities. For the mixed polygon case with multiple intersection points, only the first-in and last-out points of penetration will be recorded. The case is illustrated in Figure 4-33.

4.4 Encounter Processing

4.4.1 Mission

The Fine Filter has determined that an encounter is present for the given aircraft trajectory segment. Encounter Processing records the relevant information about the encounter in the global data base. The list of Encounters is used elsewhere in the system for the display of conflict information.

4.4.2 Design Considerations and Component Environment

This component exists to copy information from the local data base into the global data base.

Input

The input data required by Encounter Processing consists of:

- System Global Data Base
  - CURRENT_TIME

    The current system time is stored in this table.

```
ROUTINE Find_Exact_Violation_Points;
PARAMETERS
   INTERSECTION_POINTS IN,
   Volume_Id IN;
REFER TO SHARED LOCAL
   ENVIRONMENTAL_CONFLICT_DATA OUT;
DEFINE TABLES
   INTERSECTION_POINTS      The intersections grouped into the
                               enter and exit violation points
      start_time            The time associated with the
                               intersection entering the polygon
      stop_time             The time associated with the
                               intersection exiting the polygon
      begin_x               The segment on which the intersection
                               lies
      begin_y               (the first cusp)
      begin_z
      begin_t
      end_x                 (the second cusp)
      end_y
      end_z
      end_t;
DEFINE VARIABLES
   Volume_Id            The volume identifier
   Start_Time           The time of entrance violation
   Stop_Time            The time of exit violation
   Begin_T              The first cusp time
   Begin_X              The first cusp X
   Begin_Y              The first cusp Y
   Begin_Z              The first cusp altitude
   End_T                The second cusp time
   End_X                The second cusp X
   End_Y                The second cusp Y
   End_Z                The second cusp altitude
   First_In             The intersection point when the trajectory
                           first enters the polygon
   Last_Out             The intersection point when the trajectory
                           last exits the polygon
   Avg_X_Vel            The average velocity in X over the segment
   Avg_Y_Vel            The average velocity in Y over the segment
   Avg_Z_Vel            The average velocity in Z over the segment;
```

FIGURE 4-32
FIND_EXACT_VIOLATION_POINTS

```
    SELECT FIELDS start_time
       FROM INTERSECTION_POINTS
       INTO First_In
       WHERE INTERSECTION_POINTS.start_time EQ
          MIN (INTERSECTION_POINTS.start_time);
    SELECT FIELDS stop_time
       FROM INTERSECTION_POINTS
       INTO Last_Out
       WHERE INTERSECTION_POINTS.stop_time EQ
          MAX (INTERSECTION_POINTS.stop_time);
    SELECT FIELDS begin_x,begin_y,begin_z,begin_t,
       end_x,end_y,end_z,end_t
       FROM INTERSECTION_POINTS
       INTO Begin_X,Begin_Y,Begin_Z,Begin_T,End_X,End_Y,End_Z,End_T
       WHERE INTERSECTION_POINTS.start_time EQ First_Time;
    Avg_X_Vel = (End_X - Begin_X) / (End_T - Begin_T);
    Avg_Y_Vel = (End_Y - Begin_Y) / (End_T - Begin_T);
    Avg_Z_Vel = (End_Z - Begin_Z) / (End_T - Begin_T);
    X = Begin_X + Avg_X_Vel * (Start_T - Begin_T);
    Y = Begin_Y + Avg_Y_Vel * (Start_T - Begin_T);
    Z = Begin_Z + Avg_Z_Vel * (Start_T - Begin_T);
    INSERT INTO ENVIRONMENTAL_CONFLICT_DATA
       (time = Start_Time, x = X, y = Y, altitude = Z,
        volume_id = Volume_Id);
    SELECT FIELDS begin_x,begin_y,begin_z,begin_t,
       end_x,end_y,end_z,end_t
       FROM INTERSECTION_POINTS
       INTO Begin_X,Begin_Y,Begin_Z,Begin_T,End_X,End_Y,End_Z,End_T
       WHERE INTERSECTION_POINTS.stop_time EQ Stop_Time;
    Avg_X_Vel = (End_X - Begin_X) / (End_T - Begin_T);
    Avg_Y_Vel = (End_Y - Begin_Y) / (End_T - Begin_T);
    Avg_Z_Vel = (End_Z - Begin_Z) / (End_T - Begin_T);
    X = Begin_X + Avg_X_Vel * (Stop_T - Begin_T);
    Y = Begin_Y + Avg_Y_Vel * (Stop_T - Begin_T);
    Z = Begin_Z + Avg_Z_Vel * (Stop_T - Begin_T);
    INSERT INTO ENVIRONMENTAL_CONFLICT_DATA
       (time = Stop_Time, x = X, y = Y, altitude = Z,
        volume_id = Volume_Id);
END Find_Exact_Violation_Points;
```

FIGURE 4-32 (Concluded)
FIND_EXACT_VIOLATION_POINTS

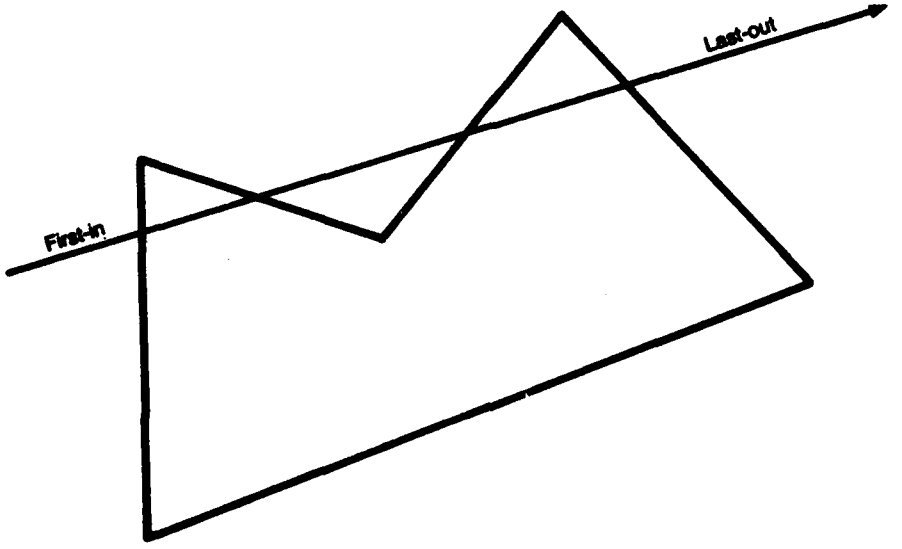


FIGURE 4-33
FIRST-IN AND LAST-OUT SELECTION

- Shared Local Data Base

  - ENVIRONMENTAL_CONFLICT_DATA

    Information stored locally which includes the enter and exit positions of the trajectory with respect to each penetration. Altitudes and times are also given.

Output

Encounter Processing updates the global data base to include:

- System Global Data Base

  - ENVIRONMENTAL_CONFLICTS

    Encounter information is stored for access by other system functions.

4.4.3 Component Design Logic

Encounter_Processing (Figure 4-34) is essentially a "housekeeping" function used to record the encounters found for a given aircraft. The data recorded in the table ENVIRONMENTAL_CONFLICT_DATA by the Fine Filter is used. The time that the system should display this predicted penetration to the cognizant controller is given as "now."

```
ROUTINE Encounter_Processing;
PARAMETERS
   Loc_Fl_Id;              The local Flight Identifier
REFER TO SHARED LOCAL
   ENVIRONMENTAL_CONFLICT_DATA IN;
REFER TO GLOBAL
   CURRENT_TIME IN,
   ENVIRONMENTAL_CONFLICTS OUT;
DECLARE VARIABLES
   Loc_Fl_Id               The local Flight Identifier;
##
   REPEAT FOR EACH ENVIRONMENTAL_CONFLICT_DATA RECORD;
      INSERT INTO ENVIRONMENTAL_CONFLICT
         (fl_id = Loc_Fl_Id,
          time = ENVIRONMENTAL_CONFLICT_DATA.time,
          x = ENVIRONMENTAL_CONFLICT_DATA.x,
          y = ENVIRONMENTAL_CONFLICT_DATA.y,
          altitude = ENVIRONMENTAL_CONFLICT_DATA.altitude,
          volume_id = ENVIRONMENTAL_CONFLICT_DATA.volume_id,
          display_as_advisory_time = CURRENT_TIME.time);
END Encounter_Processing;
```

FIGURE 4-34
ENCOUNTER_PROCESSING

APPENDIX A

AIRSPACE PROBE DATA TYPES

FL_CUSPS

TIME	x	y	z

cusp AGGREGATE (time,x,y,z)

This table contains the cusps associated with the trajectory being examined.

TIME The time associated with the cusp point

x The x coordinate of the cusp point

y The y coordinate of the cusp point

z The z coordinate of the cusp point

SEGMENTS

BEGIN_TIME	begin_x	begin_y	begin_z

end_time	end_x	end_y	end_z

```
begin AGGREGATE (begin_time,begin_x,begin_y,begin_z)
end AGGREGATE (end_time,end_x,end_y,end_z)
pair AGGREGATE (begin_time,begin_x,begin_y,begin_z,end_time,
                end_x,end_y,end_z)
```

This table contains the trajectory segments associated with the current trajectory being examined.

BEGIN_TIME The time associated with the cusp at the beginning of the segment.

begin_x The x coordinate associated with the cusp at the beginning of the segment.

begin_y The y coordinate associated with the cusp at the beginning of the segment.

begin_z The z coordinate associated with the cusp at the beginning of the segment.

end_time The time associated with the cusp at the end of the segment.

end_x The x coordinate associated with the cusp at the end of the segment.

end_y The y coordinate associated with the cusp at the end of the segment.

end_z The z coordinate associated with the cusp at the end of the segment.

FIRST_ORDER_NOMINEES

VOLUME_ID	first_cusp_time

all AGGREGATE (volume_id,first_cusp_time)

This table contains the volumes which have passed the First Order Coarse Filter.

VOLUME_ID Identifier of a volume.

first_cusp_time The time associated with the cusp known to be close to the volume.

SECOND_ORDER_NOMINEES

VOLUME_ID	first_cusp_time

all AGGREGATE (volume_id,first_cusp_time)

This table contains the volumes which have passed the Second Order Coarse Filter.

VOLUME_ID — Identifier of a volume.

first_cusp_time — The time associated with the cusp known to be close to the volume.

ENVIRONMENTAL_CONFLICT_DATA

VOLUME_ID	TIME	x	y	altitude

This table contains information on environmental conflicts for the current trajectory being examined.

VOLUME_ID — The identifier of the volume with which the environmental conflict occurred

TIME — The time associated with the environmental conflict

x — The x coordinate associated with the environmental conflict

y — The y coordinate associated with the environmental conflict

altitude — The z coordinate associated with the environmental conflict

APPENDIX B

AIRSPACE PROBE ALGORITHMS

This Appendix presents the detailed Airspace Probe elements referred to by the four Airspace Probe components. Those elements used for the determination of horizontal penetrations are found in Appendix C. Those elements are segregated to emphasize the close correlation with Appendix C of Reference 8. All other elements are listed below.

Intersection checks are performed using a linear discriminant. The discriminant is used to discriminate between points on the left side of a line and the points on the right (we may interpret the line as having a direction). This technique may be used to find which side of a trajectory segment the points of the rectangle lie. If all points lie only to one side, the segment does not intersect the rectangle. If points lie on both the left and right side, an intersection must occur (see Figure B-1).

B.1 Grid

This routine is responsible for accepting an input (x,y) position and finding the grid cell that the point is in. The Cell Id of the grid cell is returned.

```
ROUTINE Grid;
PARAMETERS
   X IN,
   Y IN,
   Box OUT;
REFER TO GLOBAL
   ENVIRONMENTAL_CELL_DIMENSIONS;
DEFINE VARIABLES
   X,                    The X coordinate of the point
   Y,                    The Y coordinate of the point
   Box;                  The cell which the point (X,Y) is in
```

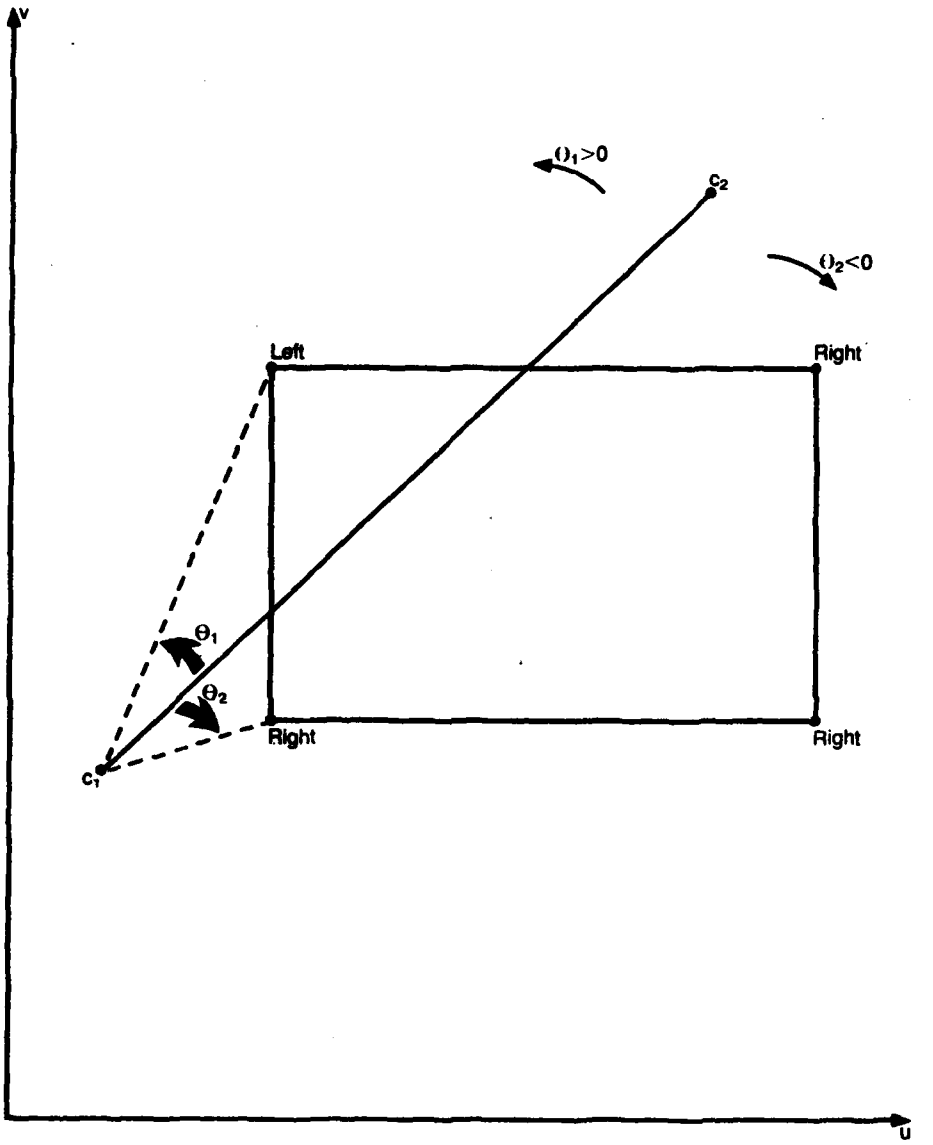


FIGURE B-1
DETERMINATION OF ORIENTATION OF A POINT TO A LINE

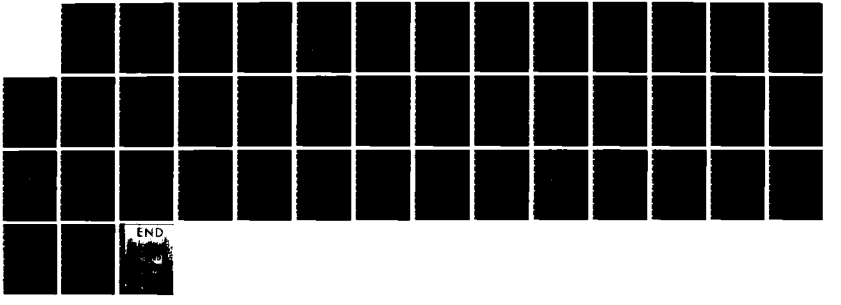
```
##
   SELECT FIELDS cell_id
      FROM ENVIRONMENTAL_CELL_DIMENSIONS
      INTO Box
      WHERE ENVIRONMENTAL_CELL_DIMENSIONS.min_x LE X AND
            ENVIRONMENTAL_CELL_DIMENSIONS.max_x GT X AND
            ENVIRONMENTAL_CELL_DIMENSIONS.min_y LE Y AND
            ENVIRONMENTAL_CELL_DIMENSIONS.max_y GT Y;
END Grid;
```

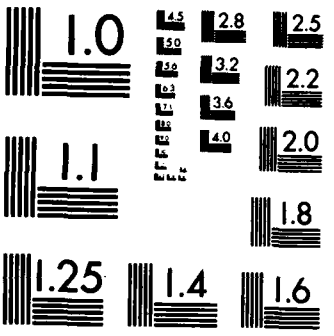
B.2 Linear Discriminant Classifier

This routine uses the coordinates of the endpoints of a line segment and the coordinates of a third point to determine which side of the line segment (left or right as measured from the first point to the second point) the third point is on. The method involves the determinant of a two-dimensional matrix whose elements are composed of the differences between the line points and the third point.

```
ROUTINE Linear_Discriminant_Classifier;
PARAMETERS
   U1 IN,
   U2 IN,
   V1 IN,
   V2 IN,
   Up IN,
   Vp IN,
   Side OUT;
DEFINE VARIABLES
   U1              The U coordinate of the first point on the line
   U2              The U coordinate of the second point on the line
   V1              The V coordinate of the first point on the line
   V2              The V coordinate of the second point on the line
   Up              The U coordinate of the point to be classified
   Vp              The V coordinate of the point to be classified
   Side            The side of the line on which the point "p" lies
   Discriminant    The value of the discriminant
##
   Discriminant = (U2 - U1) * (Vp - V1) - (Up - U1) * (V2 - V1);
   IF Discriminant GT 0
   THEN
      Side = "left"
   ELSE
      Side = "right";
END Linear_Discriminant_Classifier;
```

AD-A136 850 AUTOMATED EN ROUTE AIR TRAFFIC CONTROL ALGORITHMIC 2/2
SPECIFICATIONS VOLUME 2.. (U) FEDERAL AVIATION
ADMINISTRATION WASHINGTON DC SYSTEMS ENGINEE..
UNCLASSIFIED J A KINGSBURY ET AL. 31 SEP 83 F/G 9/2 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

B.3 Find Polygon Boundary Intersections

This routine will accept a line segment and a set of vertices representing a polygon and determine the horizontal intersection points (if there are any). The returned information is a table containing the intersection points of the segment with the polygon.

```
ROUTINE Find_Polygon_Boundary_Intersections;
PARAMETERS
   SEGMENT IN,
   Volume_Id IN,
   ORIENTATIONS OUT,
   SEGMENT_INTERSECTION_POINTS INOUT;
REFER TO GLOBAL
   VOLUME_COORDINATES IN;
DEFINE TABLES
   SEGMENT                  The current trajectory segment
      x1                    The first cusp point of the segment
      y1
      z1
      t1
      x2                    The second cusp point of the segment
      y2
      z2
      t2
      begin AGGREGATE (x1,y1)
      end AGGREGATE (x2,y2)
   SEGMENT_INTERSECTION_POINTS   The intersections with the polygon
      time                       The intersection time
      type                       The intersection location
                                    "boundary" of "interior"
      last_cusp_time             The time of the last cusp before
                                    the intersection
   ORIENTATIONS            The orientation of the cusps (IN or OUT)
      begin_orient         The orientation of the first cusp
      begin_time           The time of the first cusp
      end_orient           The orientation of the end cusp
      end_time             The time of the end cusp
      time                 The time to violation
```

```
   POLYGON_VERTICES      The vertices of the polygon
      x                  The x coordinate of the vertex
      y                  The y coordinate of the vertex
      vertex_number      The sequence number of the vertex
   PV                    The previous vertex point
      x                  The x coordinate of the vertex
      y                  The y coordinate of the vertex
   CV                    The current vertex point
      x                  The x coordinate of the vertex
      y                  The y coordinate of the vertex
DEFINE VARIABLES
   Volume_Id             The volume identifier
   C_Side                The orientation of the current vertex
   P_Side                The orientation of the previous vertex
   Begin_Side            The orientation of the first cusp point
   End_Side              The orientation of the second cusp point
   Order                 The sequence number of the current vertex
   Violation_Time        The time to violation
   Int_Count             The number of intersections thus far
   Iosum_Counter;        The number of IN/OUT intersections
   ##
   POLYGON_VERTICES = SELECT FIELDS x,y,vertex_number
      FROM VOLUME_COORDINATES
      WHERE VOLUME_COORDINATES.volume_id EQ Volume_Id;
   PV = SELECT FIELDS x,y
      FROM POLYGON_VERTICES
      WHERE POLYGON_VERTICES.vertex_number EQ 1;
   Order = 2;
   Int_Count = 0;
   Iosum_Counter = 0;
   CALL Linear_Discriminant_Classifier (S.begin IN, S.end IN,
      PV IN, P_Side OUT);
   REPEAT FOR EACH POLYGON_VERTICES RECORD;
      WHERE POLYGON_VERTICES.vertex_number NE 1 AND Int_Count LT 2;
      CV= SELECT FIELDS x,y
         FROM POLYGON_VERTICES
         WHERE POLGON_VERTICES.vertex_number EQ Order;
      CALL Linear_Discriminant_Classifier (S.begin IN,
         S.end IN, CV IN, C_Side OUT);
```

```
      IF C_Side NE P_Side
      THEN
         CALL Linear_Discriminant_Classifier (PV IN, CV IN,
            S.begin IN, Begin_Side OUT);
         CALL Linear_Discriminant_Classifier (PV IN, CV IN,
            S.end IN, End_Side OUT);
         CHOOSE CASE
            WHEN Begin_Side EQ "in" AND End_Side EQ "in" THEN
               Iosum_Counter = Iosum_Counter + 1;
            WHEN Begin_Side EQ "out" AND End_Side EQ "out" THEN
               Iosum_Counter = Iosum_Counter - 1;
            OTHERWISE
               CALL Time_To_Violation (PV IN, CV IN, S.begin IN,
                  S.end IN, Violation_Time OUT);
               INSERT INTO SEGMENT_INTERSECTION_POINTS
                  (time = Violation_Time, type = "boundary",
                  last_cusp_time = S.begin_t);
               INSERT INTO ORIENTATIONS
                  (begin_orient = Begin_Side, begin_time = S.begin_t,
                  end_orient = End_Side, end_time = S.end_t,
                  time = Violation_Time);
               Int_Count = Int_Count + 1;
      LAST_VERTEX = SELECT FIELDS ALL
         FROM THIS_VERTEX;
      Order = Order + 1;
END Find_Polygon_Boundary_Intersections;
```

B.4 Time To Violation

This routine determines the time on a given trajectory segment that a violation occurs.

```
ROUTINE Time_To_Violation;
PARAMETERS
   N1x IN,
   N1y IN,
   N2x IN,
   N2y IN,
   Cx IN,
   Cy IN,
   Ct IN,
   Px IN,
   Py IN,
   Pt IN,
   Tv OUT;
```

```
DEFINE TABLES
   VN                The vector from points N1 to N2
     x
     y
   VC                The vector from points N1 to C
     x
     y
   VP                The vector from points C to P
     x
     y
DEFINE VARIABLES
   N1x               The x value of the point N1
   N1y               The y value of the point N1
   N2x               The x value of the point N2
   N2y               The y value of the point N2
   Cx                The x value of the point C
   Cy                The y value of the point C
   Ct                The t value of the point C
   Px                The x value of the point P
   Py                The y value of the point P
   Pt                The t value of the point P
   Tp                The time from point C to P
   NXC               The cross product of N with C
   NXP               The cross product of N with C
   Tv;               The time to the violation
##
   VN.x = N2x - N1x;
   VN.y = N2y - N1y;
   VC.x = Cx - N1x;
   VC.y = Cy - N1y;
   VP.x = Px - Cx;
   VP.y = Py - Cy;
   Tp = Pt - Ct;
   NXC = CROSS(VN,VC);
   NXP = CROSS(VN,VP);
   Tv = (NXP * Tp)/(NXC + NXP);
END Time_To_Violation;
```

APPENDIX C

POLYGON HORIZONTAL VIOLATION DETERMINATION

This Appendix was taken mainly from the NAS E-MSAW Computer Program Functional Specification (CPFS) [9] and is segregated from the rest of the Airspace Probe Specification to emphasize that fact. The algorithms have been modified to account for the strategic nature of the trajectory/polygon conflicts.

An aircraft trajectory is determined to be in penetration with a polygon if any portion of any trajectory segment penetrates the adapted volume of airspace.

Due to the increased complexity of possible shapes when polygon sides are adapted such that concave angles are formed, the following procedure will be divided into two separate algorithms to facilitate the handling of the simpler (and possibly more frequent) geometries. The individual configurations considered by each algorithm are as follows:

- Algorithm 1 will operate on polygons that contain only convex angles.
- Algorithm 2 will operate on polygons with a mixture of concave and convex angles.

An indication of which algorithm is applicable to which polygons is derived by Polygon Adaptation.

C.1 Known Quantities and Relationships

The trajectory segment will be defined by the following quantities:

- Initial cusp: $C_i = (x,y,z,t)_i$
- Next cusp: $C_{i+1} = (x,y,z,t)_{i+1}$

The polygon is defined by the following adaptation derived data:

- Algorithm: Indication of which algorithm applies to this polygon
- Altitudes: Minimum and Maximum
- Total Lines: N (the total number of polygon line segments)
- Vertices: $(X_1,Y_1),(X_2,Y_2) \ldots (X_N,Y_N)$

The polygon vertices which make up the N line segments are defined in a clockwise direction. This consistent ordering is necessary since the algorithms assume that the polygon lies to the right of line segments defined by the vertices. Counterclockwise ordering (only) may be used with the proper changes in the algorithms.

C.2 Linear Discriminant Classifier

A concept essential to the understanding of the algorithms, and a computation used frequently by them, is the orientation of a point to a line. The orientation will be determined by considering an infinite line defined by the points $N_1 = (U_1,V_1)$ and $N_2 = (U_2, V_2)$ and a vector $\vec{N}$, defined from point N_1 to N_2 as follows:

$$\vec{N} = (U_2-U_1,V_2-V_1)$$

Consider also a vector $\vec{P}$, from point N_1 to point $p = (U_p, V_p)$ as (see Figure C-1):

$$\vec{P} = (U_p-U_1,V_p-V_1)$$

An expression for sin θ can be obtained by taking the cross product from $\vec{N}$ to $\vec{P}$:

$$\vec{N} \times \vec{P} = |\vec{N}|\ |\vec{P}| \sin\theta = (U_2-U_1)(V_p-V_1)-(U_p-U_1)(V_2-V_1) \qquad [C-1]$$

Since the magnitudes of vectors $\vec{N}$ and $\vec{P}$ are always nonnegative, the sign of sin θ is positive if:

$$(U_2-U_1)(V_p-V_1) - (U_p-U_1)(V_2-V_1) \geq 0.*$$

Note that if the above expression is true, the point p is confined to the area to the left of the line (as shown in Figure C-1) or is on the line. This situation will define a "left" (or "OUT") orientation of p to line N_1N_2. If the above expression is false, then the sign of sin is less than zero and p is to the right of the line. This will define a "right" (or "IN") orientation of p to the line.

---

*Note: This form corresponds to Ax mentioned in Appendix B.

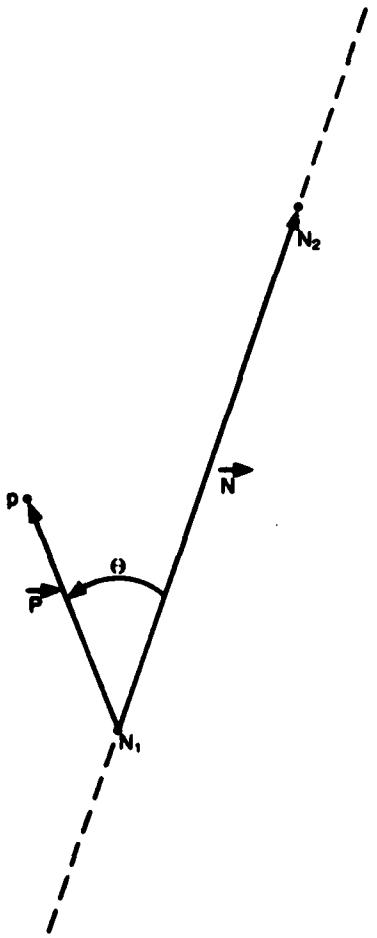


FIGURE C-1
POINT/LINE ORIENTATION

Note further that if the orientation of one endpoint of a segment is "right" and the other endpoint is "left" then the segment must cross the line at some point, but not necessarily between N_1 and N_2.

C.3 Time to Penetration

T_V will be determined as follows:

Consider the example in Figure C-2 where the trajectory segment is defined by its Cusps, $C = (U_c, V_c)$ and $P = (U_p, V_p)$ and, the polygon side is determined by its endpoints, $N_1 = (U_1,V_1)$ and $N_2 = (U_2,V_2)$. Note also that $T_p = t_p - t_c$ is the time to travel between the cusps.

The time to intersection is defined with respect to the distance to the intersection point, d, as:

$$T_V = -\frac{d}{S} \qquad [C-2]$$

or

$$d = |S|\ T_V$$

where S is the speed along the segment.

The total distance traveled during the trajectory segment is:

$$CP = |S|\ T_p \qquad [C-3]$$

By similar triangles (see Figure C-2):

$$-\frac{d}{CP} = -\frac{a}{a+b} \qquad [C-4]$$

If Equations C-2 and C-3 are now substituted into Equation C-4, and the velocity factored out, then the following expression is obtained which relates T_V to the distances a and b:

$$\frac{T_V}{T_P} = \frac{a}{a+b}$$

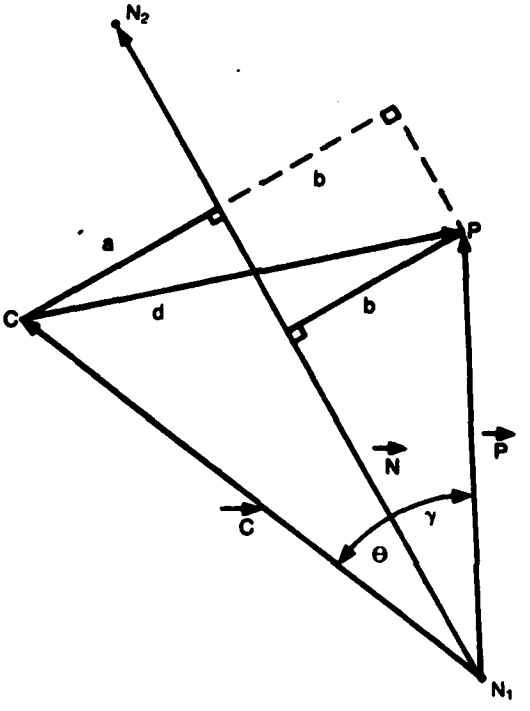


FIGURE C-2
TIME TO LINE INTERSECTION

The distance a is then determined as:*

$$a = |\vec{C}| \sin\theta$$

And the distance b as:

$$b = |\vec{P}| \sin\gamma$$

By considering the cross product of $\vec{N}$ to $\vec{C}$:

$$|\vec{C}| \sin\theta = \frac{\vec{N}\times\vec{C}}{\vec{N}}$$

And the cross product of $\vec{N}$ to $\vec{P}$:

$$|\vec{P}| \sin\gamma = \frac{\vec{N}\times\vec{P}}{\vec{N}}$$

The following relationship is obtained:

$$\frac{T_V}{T_P} = \frac{a}{a+b} = \frac{\vec{N}\times\vec{C}}{(\vec{N}\times\vec{C})+(\vec{N}\times\vec{P})}$$

T_V can now be expressed in terms of T_P and the cross products $\vec{N}\times\vec{C}$ and $\vec{N}\times\vec{P}$ as:

$$T_V = \frac{(\vec{N}\times\vec{C})T_P}{(\vec{N}\times\vec{C})+(\vec{N}\times\vec{P})}$$

C.4 Convex Polygon Intersections

The Convex_Polygon_Intersection_Check (Figure 4-26) operates on polygons that contain only convex angles between sides. This algorithm is very similar to Mixed_Polygon_Intersection_Check (Figure 4-27) which is a more general algorithm. The convex case is discussed separately here, since certain efficiencies (not addressed here) can be incorporated to enhance the performance of the algorithm on convex polygons. The algorithm loops through the sides of

*Angles are measured in a counter-clockwise direction.

the polygon to determine if intersections exist. A possible intersection is noted if the orientation of a vertex does not match the orientation of the previous vertex (sequencing in a clockwise fashion). To find if an intersection truly exists, the orientation of the cusps with respect to the polygon side are examined. If an intersection indeed exists, then the time to penetration is calculated and recorded in the list of intersections for the given polygon/trajectory segment pair.

If no intersections occur, the algorithm checks to see if the segment is completely within the polygon. If only one intersection occurs, the algorithm checks to see which cusp is inside the polygon. In both cases the included points are added to the intersection list. See section C.5 for more discussion on inclusion/exclusion of points with respect to a polygon.

C.5 Mixed Polygon Intersections

Mixed_Polygon_Intersection_Checks (Figure 4-27) operates on polygons which contain both convex and concave angles. The algorithm will loop through the sides which define the polygon. The orientation of the polygon sides to the trajectory segment will be examined to determine whether a penetration is possible.

To gain an understanding of how the mixed algorithm operates, a few points that must be assumed will be presented.

- If any polygon is crossed by an "infinite" length line, the "ends" of that line are outside the polygon area (see Figure C-3).

- As a point moves along this line, each time it crosses a polygon side its state is altered. Its state varies between IN or OUT of the polygon area.

- An "infinite" line crossing any polygon will cross an even number of sides. Since such a line begins outside of the polygon and ends outside of the polygon, an even number of crosses must have occurred.

- If a point is within a polygon area, and an "infinite" line is laid over the point, the point would cross an odd number of sides if the point moved to either end of the line. This is because its state has been altered from IN to OUT.

- If a point is outside the polygon and an "infinite" line is laid over the point, the point would cross an even number of sides (or no sides) as it moved to either end. This is true because its state (OUT) has not changed.

It is visually easy to see if a point is IN or OUT by counting the sides crossed as it moves towards an end, but computationally difficult. All sides must be searched to see if they are crossed by the line and if they lie between the point and the chosen end.

The mixed algorithm uses the above information in a slightly different form. Instead of a point, the trajectory segment is used and an infinite vector is laid over it (see Figure C-4).

The infinite vector is called the trajectory path. In most cases the mixed algorithm must search all sides to see if they have been crossed by the trajectory path. If a side is crossed by the path, a cross product is employed to see if the trajectory path is IN or OUT relative to the particular side. (If the trajectory path actually entered at the side, it would instantly be known that part of the path is inside the polygon.)

To relate this back to the idea of moving from a point to the end of an infinite line (see Figure C-4), an IN orientation would mean that the moving point was IN the polygon area before it crossed the side moving towards an end.

The mixed algorithm keeps a running sum of the INs and OUTs, where IN = +1 and OUT = -1. A final sum of 0 means that an even number of sides were crossed between the trajectory segment and either end of the trajectory path. Therefore, a sum of 0 means that the trajectory segment was entirely outside the polygon area. A final sum of +2 means that an odd number of sides were crossed in either direction and the entire trajectory segment is therefore inside the polygon area (see Figure C-5).

Special situations which can occur are as follows:

- The trajectory path coincides with a polygon vertex, but a moving point passing through the vertex would not alter its state of being IN or OUT (see Figure C-6). In Figure C-6a, vertex V_a coincides with the trajectory path. Since a point moving through this vertex would always remain outside the polygon, the running sum should not change. In Figure C-6b we have the same situation, but the point never varies from being inside the polygon as it passes through vertex V_b.

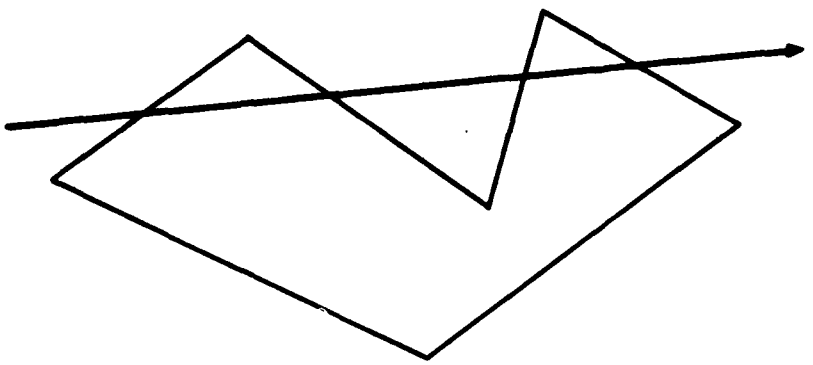


FIGURE C-3
INFINITE LINE CROSSING A MIXED POLYGON

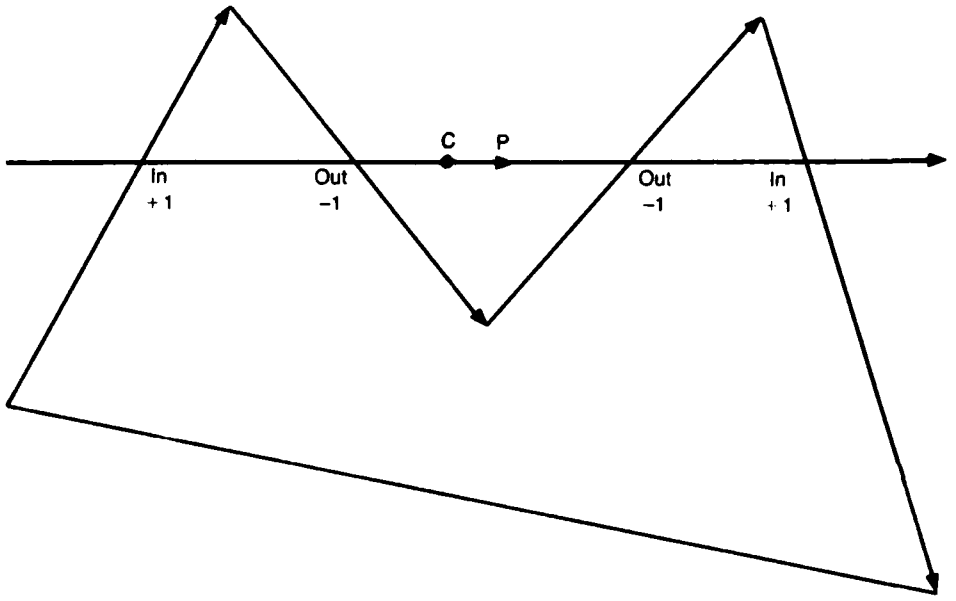


FIGURE C-4
MIXED POLYGON AND TRAJECTORY SEGMENT OUTSIDE

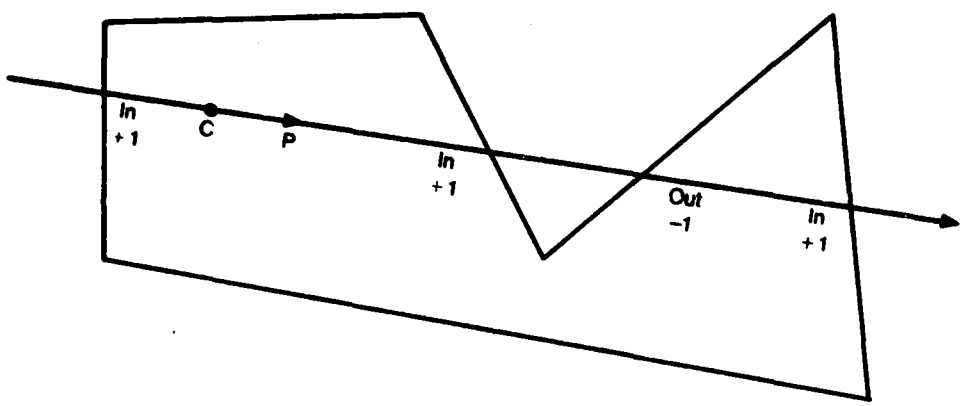
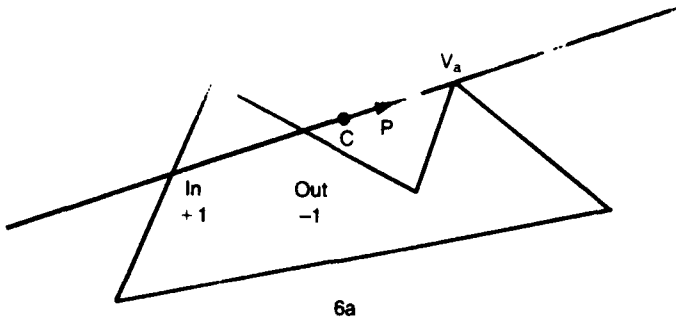
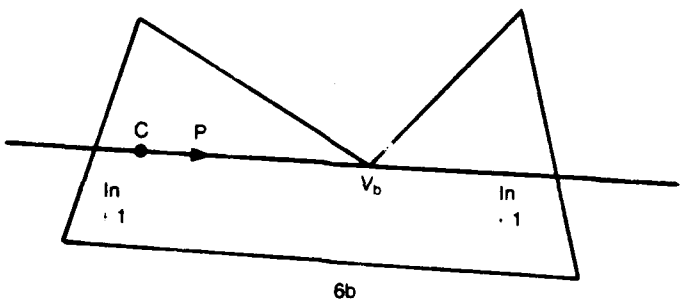


FIGURE C-5
MIXED POLYGON AND TRAJECTORY SEGMENT INSIDE



6a



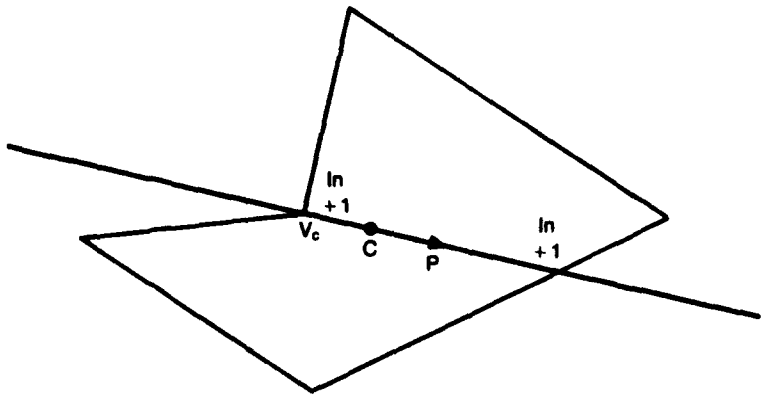
6b

FIGURE C-6
MIXED POLYGON SPECIAL SITUATION 1

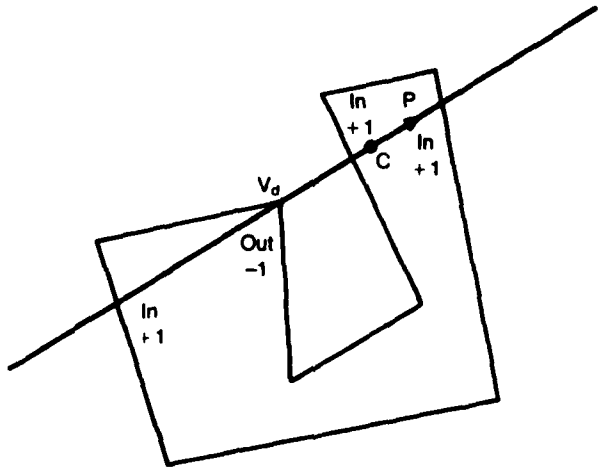
- The trajectory path coincides with a polygon vertex, but a moving point's state would be altered as it crossed over the vertex (see Figure C-7). In Figure C-7a, at vertex V_c, a +1 should be added to the running sum. In Figure C-7b, at vertex Vd, a -1 should be added to the sum.

- The trajectory path coincides with a side of the polygon, and like situation 1, the running sum should not change (see Figure C-8).

- The trajectory path coincides with a side of the polygon, and like situation 2, the running sum should change (see Figure C-9).

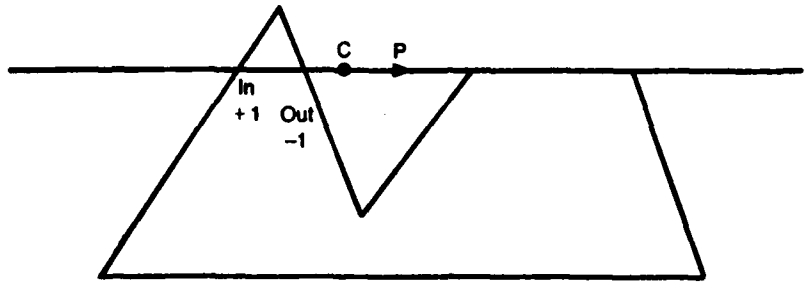


(a)

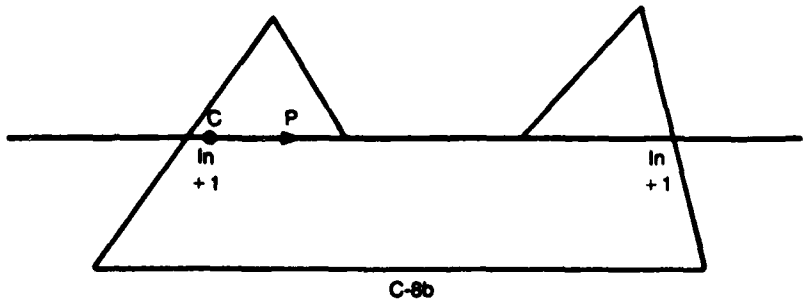


(b)

FIGURE C-7
MIXED POLYGON SPECIAL SITUATION 2

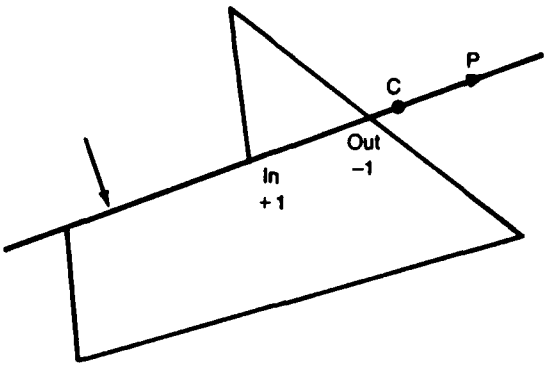


C-8a



C-8b

FIGURE C-8
MIXED POLYGON SPECIAL SITUATION 3



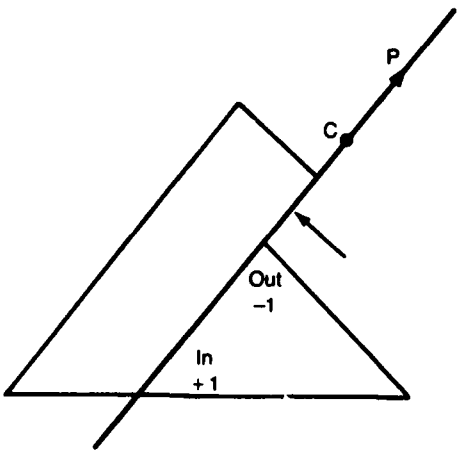


FIGURE C-9
MIXED POLYGON SPECIAL SITUATION 4

APPENDIX D

GLOSSARY

Numbers in parenthesis at the end of the definition refer to the section in which the term is first used.

AAS	Advanced Automation System (1.1).
Adaptation	The process of collecting environmental data and storing it in system data bases (1.5.1).
Along Route Distance	The distance of a converted fix on the route from the first converted fix (2.1.1).
AERA	The concept of automated en route air traffic control described in "The AERA Concept" [12] (3.4).
AGD Variable	An AGD variable is an element (gradient, direction or acceleration) of the AGD Vector (2.1.3). (See also "AGD Vector")
AGD Vector	The AGD vector is the 3-tuple (acceleration, gradient and direction) controlling the construction of a segment (2.1.3).
Air Traffic Controller	See "Controller" (1.4.1).
Area	An area is a second level division of the continental United States Airspace. Controllers are specially trained for an area's airspace, a region bounded horizontally by a polygon and having some vertical extent (1.4.1). (See also "Center" and "Sector")
ARTCC	Air Route Traffic Control Center (1.4.1). (See also"Center")
ATC	Air Traffic Control (1.1).
Cell	A discrete compartment of the wind grid (2.1.1).

Center	A center is the administrative headquarters and the operational facility for control of the first-level division of the Continental United States Airspace. The center controls a region bounded horizontally by a polygon and vertically by the Center floor and an altitude of 60,000 feet (1.4.1). (See also "Area" and "Sector")
Clearance	**A specially formatted order from the controller to the pilot which commands the pilot to execute a maneuver (2.1.3).**
Component	**Third-level algorithmic unit in the breakdown of an automation function (1.3). (See also "Subfunction" "Element")**
Controller	**An en route radar controller as defined in (1.4.1).**
Converted Fix	**A fix that is a component of the aircraft route after Route Conversion processing (1.4.1.2). (See also "Fix" and "Coordination Fix")**
Converted Route	The filed route of flight as augmented in Route Conversion with preferred arrival routes, among others (1.5.2).
Coordination Fix	**A special purpose fix used for a reference location when flight plans are transmitted to the next control area (1.5.2). (See also "Fix" and "Converted Fix")**
Cusp	**An aircraft trajectory is represented as a series of points called cusps. The cusps are the points of possible AGD vector discontinuity (2.1.2).**
Element	**Fourth-level algorithmic unit in breakdown of an automation function (1.3). (See also "Subfunction" and "Component")**
FAA	**Federal Aviation Administration (1.1).**
Fix	**A named x,y location (1.4.1.2).**
Grid Cells	**Discrete compartments of the wind grid (2.1.1).**

Man-Machine Interface	Interaction mechanism provided by the computer system to translate human input into internal format and translate internal format into human readable form (2.1.2).
NAS	National Airspace System (1.1).
Next Cusp	The next position to which the aircrft route will be modeled (2.1.2).
Past Cusp	The position to which the aircraft route has been modeled (2.1.2). (This point may be at some future position in terms of the current actual aircraft position.)
PDL	Program Design Language (1.2 and Appendix E).
Pending Action List	A list which contains planned actions which may effect the aircraft trajectory from the past cusp onward (2.1.3). (See also "Past Cusp" and "Planned Action")
Plan	A set of planned actions for an aircraft (1.5.2). (See also the definition of "Planned Action")
Planned Action	An internal representation of a proposed change of aircraft clearance which can be modeled into the aircraft trajectory (2.1.2).
Planning Region	The geographic area over which the Trajectory Estimation algorithm operates. This area includes the extent of an entire Air Route Traffic Control Center (ARTCC) and also includes a buffer area (2.1.1).
Profile Reference Point	A 4-space position used to initialize Trajectory Estimation (1.5.2).
Sector	A sector is the third level division of the Continental United States airspace. A sector is the division to which a controller is assigned (1.4.1). (See also the definition of "Center" and "Area")

Segment	A segment is a part of an aircraft trajectory represented by an implied line between two adjacent cusps. The gradient, direction, and acceleration of the aircraft are constant across the segment (2.1.2).
Stimulus	A stimulus is one of several flight path events related to a planned action which initiate the planned action processing component (2.1.3).
Subfunction	The second-level algorithmic unit in the breakdown of an automation function (1.3). (See also "Component" and "Element")
Trajectory	A description of an aircraft's position in (x,y,z,t) space, produced by applying altitude and timing assumptions to the filed flight plan and revising when necessary (1.4.1.2).
Wind Grid	A grid structure overlaid on the planning region to relate geographic coordinates to wind speed, direction and temperature at that location (2.1.1).

APPENDIX E

AERA PDL LANGUAGE REFERENCE SUMMARY

E.1 Overview of the Use of AERA PDL

The AERA Program Design Language (PDL) has been created for the single purpose of presenting algorithms in this specification document. It evolves from previous AERA uses, and from MITRE WP-81W552, "All About E," October 1981.

The description of this appendix is intended to support readers and users of AERA PDL. AERA PDL supports readable, yet structured and consistent, descriptions of algorithms.

AERA PDL Features

- Relational data tables can be defined and manipulated by constructs in the language.
- Builtin functions are used to provide routine calculations without showing all of the detail.
- Routines are used to modularize logic paths and data scope.
- Indentation is used to indicate statement grouping, statement continuation, and levels of nesting.
- Routines explicitly define data or refer to predefined data.

AERA PDL Statements

The types of statements used in AERA PDL are:

- English language statements
- assignment statements
- routine declaration statements
- data manipulation statements
- flow of control statements

E.2 Elements of AERA PDL

Keywords

Keywords are words reserved for the usage of AERA PDL. Figure E-1 presents all the keywords used in the current version of AERA PDL, grouped for convenience.

routine construction keywords

CALL	END	ROUTINE

data reference keywords

PARAMETERS	IN
REFER TO GLOBAL	OUT
REFER TO SHARED LOCAL	INOUT
DEFINED IN GLOSSARY	

data definition keywords

DEFINE CONSTANT(S)
DEFINE VARIABLE(S)
DEFINE TABLE(S)

common arithmetic builtin function keywords

AVG	MIN	ABS	EXP	COS	ARCCOS
SUM	MAX	CEIL	LOG	SIN	ARCSIN
PROD	MEDIAN	FLOOR	SQRT	TAN	ARCTAN
		SIGNUM			
		MOD			

coordinate geometry builtin function keywords

DIST	DOT	INTERSECTION
MAGNITUDE	CROSS	INTERPOLATE
DIRECTION	LINE	

set builtin function keywords

UNIQUE	COUNT	CONCAT	BOOL

FIGURE E-1
KEYWORD GROUPINGS

set operator keywords

UNION INTERSECT

table manipulation keywords

SELECT FIELDS	ALL
INSERT INTO	FROM
DELETE FROM	INTO
UPDATE IN	WHERE
	ORDERED BY
	RETURN COUNT

value constant keywords

TRUE FALSE NULL

comparison keywords

NOT	GT	EQ	ANY
OR	GE	NE	ALL
AND	LT	IS IN	
	LE	IS NOT IN	

flow of control keywords

IF ... THEN ... ELSE
CHOOSE CASE ... WHEN ... THEN ... OTHERWISE
FOR ... TO
REPEAT WHILE
REPEAT UNTIL
REPEAT FOR EACH ... RECORD
GO TO

FIGURE E-1 (Concluded)
KEYWORD GROUPINGS

Operators

The operators of AERA PDL are summarized in Figure E-2.

The Assignment Operator

- The format of the assignment statement is:
  "target" = "expression"
- The target may be any type of data allowed by AERA PDL.
- The assignment operator includes the ability to fill a table from data contained in other tables. The form of this use of the assignment operator is:
  "table_name" = "table_expression" ;

Builtin Functions

The builtin functions of AERA PDL accept either an single value or data organized into an array. The author of a routine must make it clear in comments and text what form of data is being processed by the builtin function. Builtin functions are listed in Figure E-3.

E.3 Routine Construction

The order of appearance of constructs in a routine is:

- ROUTINE -- required
- PARAMETERS -- optional
- REFER TO GLOBAL -- optional
- REFER TO SHARED LOCAL -- optional
- DEFINED IN GLOSSARY -- optional
- DEFINE CONSTANTS -- optional
- DEFINE VARIABLES -- optional
- DEFINE TABLES -- optional
- logic flow -- required, but will vary by routine.
- END -- required

Three of the constructs are noted below:

The ROUTINE Construct

- The ROUTINE construct names the routine.
- The syntax of the ROUTINE construct is:
  ROUTINE "routine_name" ;

assignment operator

A = B	A is assigned the value of B

arithmetic operators

A + B	A plus B
A - B	A minus B
A * B	A times B
A / B	A divided by B
A ** B	A to the power of B

comparison operators

A LT B	A is less than B
A LE B	A is less than or equal to B
A GT B	A is greater than B
A GE B	A is greater than or equal to B
A EQ B	A is equal to B
A NE B	A is not equal to B

logical operators

NOT A	The logical opposite of A
A OR B	Logical OR of A and B
A AND B	Logical AND of A and B

set operators

A INTERSECT B	The set intersection of A and B
A UNION B	The set union of A and B
A IS IN B	A is an element of the set B
A IS NOT IN B	A is not an element of the set B

FIGURE E-2
GROUPINGS OF AERA PDL OPERATORS

FUNCTION	MEANING
ABS(x)	Absolute value of x
ARCCOS(x,y)	Inverse cosine of the ratio of y to x
ARCSIN(x,y)	Inverse sine of the ratio of y to x
ARCTAN(x,y)	Inverse tangent of the ratio of y to x
AVG(A)	Mean of the elements in A
BOOL(x)	Numerical equivalent of logical condition: 1 if x is TRUE, 0 if x is FALSE
CEIL(x)	Smallest integer greater than or equal to x
CONCAT(s1,s2,...,sN)	Concatenation of strings s1 through sN
COS(x)	Cosine of x
COUNT(A)	Number of elements of a set A
CROSS(v1,v2)	Cross product of vectors v1 and v2
DIRECTION(p1,p2)	Direction of p2 from p1 in degrees from the north; usually will be expressed in degrees clockwise from true north
DIST(p1,p2)	Euclidean distance between points p1 and p2
DOT(v1,v2)	Dot product of vectors v1 and v2
EXP(x)	e to the x power
FLOOR(x)	Greatest integer less than or equal to x

FIGURE E-3
BUILTIN FUNCTIONS

FUNCTION	MEANING
INTERPOLATE(a,b,t)	The point (1-t)a+tb
INTERSECTION(L1,L2)	The point of intersection of the lines L1 and L2
LINE(p1,p2)	Vector (a,b,c) corresponding to the line ax + by = c which passes through the points p1 and p2
LOG(x)	Log of x in base e
MAGNITUDE(v)	Length (i.e., norm) of the vector v
MAX(A)	Largest of the elements in the set A
MEDIAN(A)	Median value of the elements in set A
MIN(A)	Smallest of the values in set A
MOD(x1,x2)	Remainder when x1 is divided by x2
PROD(A)	Product of the elements in A
SIGNUM(x)	Function yielding 1 if x GT 0, -1 if x LT 0, and 0 if x EQ 0
SIN(x)	Sine of x
SQRT(x)	Square root of x
SUM(A)	Sum of the elements in A
TAN(x)	Tangent of x
UNIQUE(A)	The set A with no duplicate elements

FIGURE E-3 (Concluded)
BUILTIN FUNCTIONS

The CALL Construct

- The CALL construct invokes use of another routine as a subroutine and passes to it the data on which it is to operate.

- The syntax of the CALL construct is:
  CALL "routine_name" ("data_usage_list") ;

- The data usage list in the CALL statement must match in number and data utilization (IN, OUT, INOUT) the PARAMETERS statement of the called routine.

The END Construct

- The END construct shows the formal end of the routine.

- The syntax of the END construct is:
  END "routine_name" ;

E.4 Data Definitions

Data usage is defined in the constructs placed at the beginning of each routine.

The structures, or organization of data, recognizable to AERA PDL include constants, atomic variables, hierarchically structured variables, arrays, tables, and field-types. The hierarchically structured variables are the same as the structure variables of PL/I.

Within a table, the values corresponding to the definition of a field-type are called fields when they are referred to individually. The values for a whole column of a table (or a subset of the whole column) may be referred to as a set of fields.

The following data definition constructs appear in the order shown, if any are needed. The first line of each construct begins in column 1, aligned with the ROUTINE construct.

The PARAMETERS Construct

- This construct provides usage information about the data that are being provided by the calling routine in the form of specification of read-only 'IN', write-only 'OUT', or modification of an existing value 'INOUT'.

- Variables appearing in the PARAMETERS construct are still local data for the routine being defined and as such appear in the definition constructs.

- The syntax of the PARAMETERS construct is:
  PARAMETERS "data_usage_list" ;

The REFER TO GLOBAL Construct

- This construct provides reference to, and usage information for, data from the Global data model.

- The syntax of the REFER TO GLOBAL construct is:
  REFER TO GLOBAL "data_usage_list" ;

The REFER TO SHARED LOCAL Construct

- This construct provides reference to, and usage information for, data from the Shared Local data model described in Appendix A of the specification.

- The syntax of the shared local construct is:
  REFER TO SHARED LOCAL "data_usage_list" ;

The DEFINED IN GLOSSARY Construct

- This construct provides reference to, and usage information for, data from a specially prepared Glossary that centralizes the definition of data variables that are used repeatedly within a given function of the algorithmic specification.

- The syntax of the shared local construct is:
  DEFINED IN GLOSSARY "data_usage_list" ;

The DEFINE CONSTANTS Construct

- The use of named constants instead of in-line numerical constants is available at the discretion of the author of an algorithm. Named constants, if present, are to be declared with this construct.

- The syntax of the DEFINE CONSTANTS construct is:
  DEFINE CONSTANTS "constant_definition_block" ;

The DEFINE VARIABLES Construct

- The syntax of the DEFINE VARIABLES construct is:
  DEFINE VARIABLES "variable_definition_block" ;

The DEFINE TABLES Construct

- The syntax of the DEFINE VARIABLES construct is:
  DEFINE TABLES "table_definition_block";

E.5 Flow of Control Constructs

The IF...THEN...ELSE Construct

- The syntax of the IF...THEN...ELSE construct is:
  IF "condition"
  THEN
  "statement_block"
  [ELSE
  "statement_block"]

The CHOOSE CASE Construct

- This construct provides a choice of one of several alternative logic paths depending on the first condition satisfied among the conditions specified.

- The OTHERWISE phrase is optional.

- The syntax of the CHOOSE CASE construct is:
  CHOOSE CASE
  WHEN "condition" THEN
  "statement_block"
  [WHEN phrases repeated as necessary]
  [OTHERWISE
  "statement_block"]

The REPEAT WHILE Construct

- The syntax of the REPEAT WHILE construct is:
  REPEAT WHILE "condition" ;
  "statement_block"

The REPEAT UNTIL construct

- The syntax of the REPEAT UNTIL construct is:
  REPEAT UNTIL "condition" ;
  "statement_block"

The REPEAT FOR EACH RECORD Construct

- This construct explicitly loops over all records in table, or the subset of a table as specified in a WHERE phrase.

- The syntax of the REPEAT FOR EACH construct is:

```
REPEAT FOR EACH "table_name" RECORD
  [ WHERE "condition" ] ;
     "statement_block"
```

- Within the statement block of this loop, the construct of "table_name"."field_name" means only the ONE value that is associated with the record for that iteration of the loop.

- If it is necessary to refer to entire columns of the table that is being looped on, the correct form of the reference is ALL("table_name"."field_name"). This construct means exactly what "table_name"."field_name" would have meant if the loop had not been in effect.

The GO TO Construct

- The syntax of the GO TO construct is:

```
GO TO "label" ;
```

The FOR...TO... Construct

- The syntax of the FOR...TO... construct is:

```
FOR "loop_index" = "initial_value" TO "last_value" ;
    "statement_block"
```

E.6 Table Manipulation Constructs

The SELECT FIELDS Construct

- This construct extracts data from a table, or from a collection of tables, and makes it available to the routine.

- The syntax of the SELECT FIELDS construct is:

```
SELECT FIELDS [ UNIQUE ] [ "field_list" | ALL]
   FROM "table_name_list"
 [ INTO "local_variable_name_list" ]
 [ WHERE "condition" ]
 [ ORDERED BY "field_name" ]
 [ RETURN COUNT ( "local_variable" ) ] ;
```

The INSERT INTO Construct

- This construct allows a new record to be inserted into a table.

- The syntax of the INSERT INTO construct is:
  INSERT INTO "table_name" ("field_assignments")
  [WHERE "condition"] ;

- All insertions will preserve the assumption of no duplicate records allowed in the table.

The UPDATE IN Construct

- This construct allows existing records in a table to have certain of their values changed.

- The syntax of the UPDATE IN construct is:
  UPDATE IN "table_name" ("field_assignments")
  [WHERE "condition"] ;

The DELETE FROM Construct

- This construct removes selected records from a table.

- The syntax of the DELETE FROM construct is:
  DELETE FROM "table_name"
  [WHERE "condition"] ;

E.7 Glossary

"comparison"

- There are four possible syntaxes for the comparison. These are not given separate names, but will all be shown as if they shared the same element of the language.

- The first syntax is for arithmetic comparisons:
  "individual" GE|LE|GT|LT "individual"

- The second syntax is for general comparisons:
  "individual" EQ|NE "individual"

- Both of these syntaxes are also valid if they are used to compare two variables with the same complex organization, for example two vectors of the same length or two field types from the same table. In this case the result has as many answers as there are elements in the compared variables.

- The third syntax is for arithmetic comparisons:
  "individual" GE|LE|GT|LT ANY|ALL "set"

- The fourth syntax is for general comparisons:
  "individual" IS IN|IS NOT IN "set"

- The latter two syntaxes are used to qualify an individual based on any value in a set of values.

"condition"

- The syntax of the condition is:
  "comparison" [AND|AND NOT|OR|OR NOT "comparison"]

- The optional part of this syntax can be repeated as often as required.

"constant definition block"

- The content of the constant definition block is three columns: the constant names, the constant values, and the constant descriptions.

- The constant names are aligned in a column 3 spaces indented from the DEFINE CONSTANTS line.

- The other two columns are aligned as convenient, so that there is no visual overlap between the columns.

"data usage list"

- A routine must declare the type of use for all of its data that are known outside the routine.

- The three types of use are: read only (IN), create (OUT), and modify an existing copy (INOUT).

- The format of a data usage list is:
  "variable_name" "usage_type", ...

- An example of the format for data usage list is:
  An_Input_Parameter IN, A_LOCAL_TABLE INOUT

"expression"

- Variables may be formed implicitly in expressions without being separately named or defined.

- Expressions are combinations of defined variables with the operators and builting functions of AERA PDL.

- In an expression, the implicit variable output from any builtin function may be used as the input to any other builtin function or operator.

- An expression, when fully evaluated, yields one variable.

"field assignments"

- This term only appears in statements referring to exactly one table: INSERT and UPDATE.

- The form of the term is a comma-separated list:
  "field_assignment", ...

- The form of a single assignment is:
  "field_name" = "value_expression"

- In this term the field_names do not have to be qualified by the table name (that is given in the statement).

"table definition block"

- Three types of definition are made in this block: table definitions, field-type definitions, and AGGREGATE definitions.

- Table definition lines are formatted as:
  "table_name" "table_definition"

- Field-type definitions lines are formatted as:
  "field_name" "field_definition"

- Aggregate definitions are formatted as:
  "aggregate_name" AGGREGATE ("field_name_list")

- Fields will contain only atomic (single-valued) variables.

- Aggregates may be used so that a program can manipulate multiple fields in one statement when it makes sense to do so.

"table-expression"

- Tables may be used implicitly in assignments or comparisons being separately named or defined.

- A table expression is either a table name or a SELECT statement specifying the contents of the implicit table.

"table name"

- Generally, this is just the name of a table.

- In a few statements, there is a syntax that allows a user to define a synonym and use it in the rest of that statement. The intent of this option is to allow shorter where clauses that are easier to read. The format of the synonym reference is:
  "existing_table_name" ("synonym")

- The statements that allow this use are those that have the where clause: SELECT, INSERT, DELETE, UPDATE, and REPEAT.

"variable definition block"

- The content of the variable definition block is two columns: variable names and variable descriptions.

- Align variable names in a column that is indented 3 spaces from the DEFINE VARIABLES line.

- Align variable definitions in a column as convenient; when a structure element is defined, both the variable name and the variable definition should be indented three spaces from the name and definition of the next higher level variable.

- Three types of variables may be defined in this block: atomic variables, arrays, and structured variables.

- Each element variable is described by a line:
  "variable_name" "variable_definition"

- Each array variable is described by a line:
  "variable_name" ("dimensions") "variable_definition"

- Each structured variable is described by multiple lines, one line per lowest level element, and one line for each named level of grouping/structure, with indentation levels used to indicate the grouping.

- The names of subordinate elements of a structured variable are named in all lower case letters.

- The use of complex structured variables is not encouraged; one reasonable use for them is to receive the values of AGGREGATEs.

E.8 Other Uses and Conventions

Use of Special Characters in AERA PDL

- Parentheses are used for grouping statements and setting of special parts of the constructs.
- Semicolons are used as statement terminators.
- Colons are used to terminate labels.
- Underscore is used to separate words in multi-word identifiers.
- The symbols '+','-','*', and '/' are used as arithmetic operators.
- The pound sign '#' is used as a comment delimiter, for beginning and end of each comment line.
- Commas are used as separators in lists of operands.
- Periods are used to separate fully qualified names.

Naming Conventions

- Keyword identifiers use only uppercase letters and are underlined. They are the only underlined identifiers in the PDL.
- Table identifiers from the relational data base also use only uppercase letters.
- AGGREGATE identifiers for combinations of fields use no uppercase letters.
- References to fields in a table, used in the normal course of reference in AERA PDL, will be fully qualified by including the table name.

Other Identifiers

- Identifiers for constants, routines, labels, arrays, and hierarchically structured variables are all be named using word-initial capitals.
- For hierarchically structured variables, all of the subordinate elements within the structure use only lowercase letters.
- For hierarchically structured variables, all references to the subordinate elements in the structure will be in fully qualified form using separate identifiers.
- Global data and shared local data can include both tables and parameters. The individual parameters are named using word-initial capitals.

Use of the Formal Constructs in AERA PDL Statements

- Statements may use formal constructs or clear English descriptions to specify the intended test or action.
- Any AERA PDL statement is terminated by a semicolon, including any English statement outside of a comment.
- Below the level of statement, some statements have a finer organization in terms of "phrases", usually occupying a line per phrase and indented one level from the first line of the original statement.

Statement Organization

- Indentation is used to indicate statement grouping, statement continuation, and levels of nesting.
- Any statement may have a label starting in column 1.
- Continuation lines are indented three spaces from the original line of the statement.
- Comments are used as needed, bracketed by the special character '#'.

APPENDIX F

REFERENCES

1. Advanced Automation System: System Level Specification, U.S. Department of Transportation, Federal Aviation Administration, FAA-ER-130-005B, April 1983.

2. U.S. Department of Transportation, Federal Aviation Administration, National Airspace System Plan: Facilities, Equipment, and Associated Development, April 1983.

3. W. J. Swedish et al., "Operational and Functional Description of AERA 1.01," MTR-83W69, The MITRE Corporation, McLean, Virginia, May 1983.

4. U.S. Department of Transportation, Engineering & Development Major Program Plan, Federal Aviation Administration, August 5, 1982.

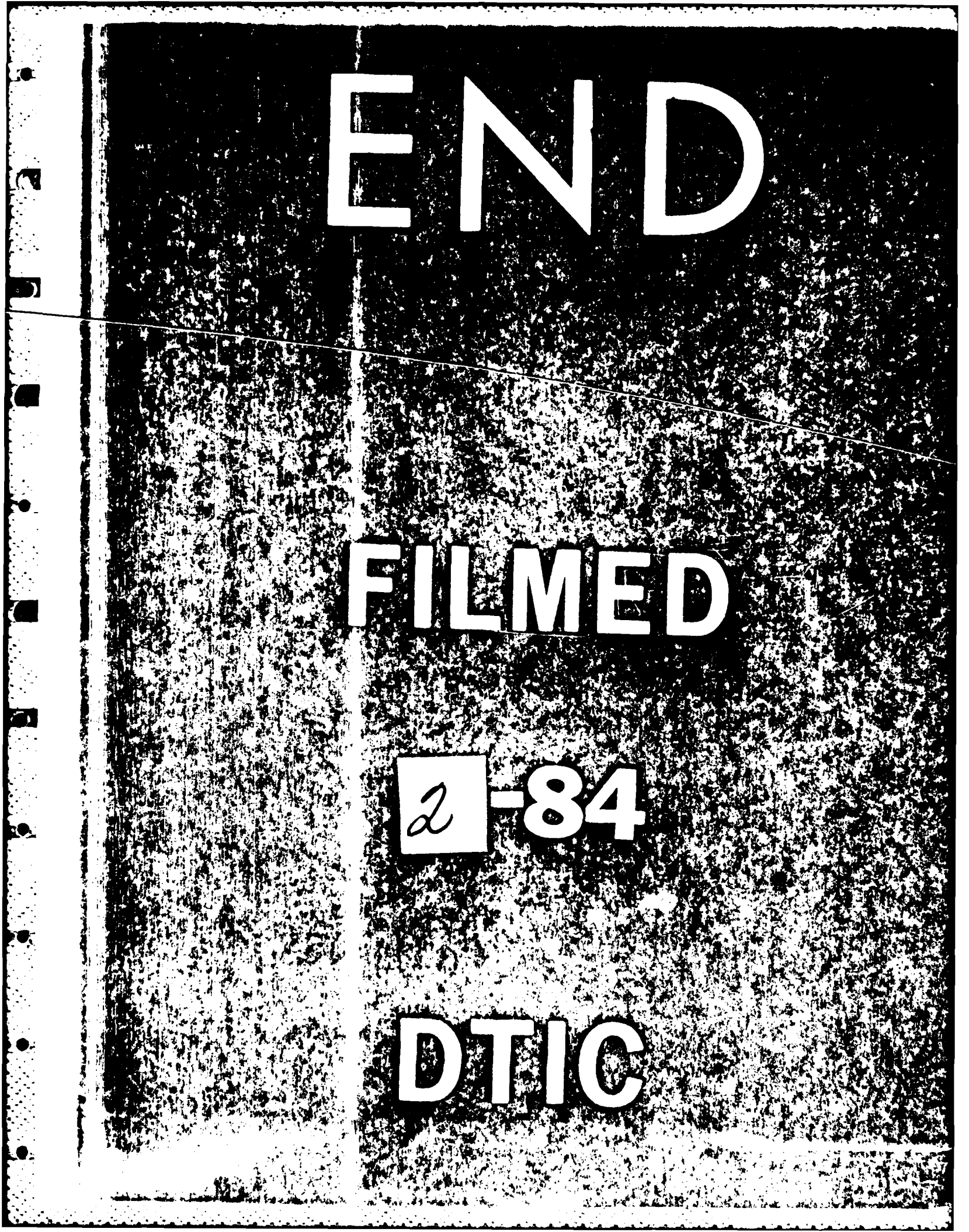
5. W. J. Swedish, "Evolution of Advanced ATC Automation Functions," WP-83W149, The MITRE Corporation, McLean, Virginia, March 1983.

6. U.S. Department of Transportation, Federal Aviation Administration, Airman's Information Manual, Basic Flight Information and ATC Procedures, January, 1982.

7. U.S. Department of Transportation, Federal Aviation Administration, National Airspace System Configuration Management Document, NAS-MD-326, August 1, 1982.

8. U.S. Department of Transportation, Federal Aviation Administration, National Airspace System Configuration Management Document, Automated Tracking, NAS-MD-321, August 1, 1982.

9. "Preliminary Computer Program Functional Specification (CPFS) for En Route-Minimum Safe Altitude Warning (E-MSAW)," Implementation, DOT-FA76WA-3815, April 1979.


END
FILMED
2-84
DTIC